BlueGene/L Consortium

System Software Workshop

Pete Beckman

Feb 23 – 24, Salt Lake City, Utah

Agenda Notes: Wed

- 08:00 Welcome and Introduction
- 08:30 Tech Session 1
- 10:30 Break
- 11:00 Tech Session 2
- 12:30 Lunch on your own
- 01:30 Tech Session 3
- 03:30 Break
- 03:45 Tech Session 4
- 05:00 Dismiss
- 06:00 Reception
- 08:00 Dinner on your own

Agenda Notes: Thurs

- 08:30 Tech Session 1
- 10:30 Break
- 11:00 Tech Session 2
- 12:30 Lunch on your own
- 01:30 Tech Session 3
  - Futures Round Table
  - BG Consortium, Purchasing, Access
- 03:00 Dismiss

Special Thanks

Goals for Workshop

- Learn about the system software currently available and in development for BG/L
- Share experiences and solutions
- Prioritize system software enhancements and requirements
- Find areas where community Open Source development can extend the BG/L environment
- Organize community efforts and build collaboration

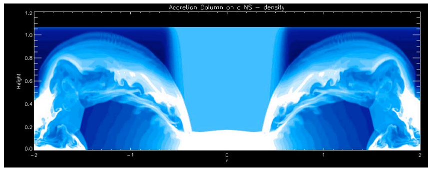
The Road To Petaflops

- Petaflop computing is just a couple years away, and BlueGene-like architectures will lead the way
- What **didn't** we do to get here?
  - Develop new exotic automatically parallelizing compilers
  - Develop new multi-threaded functional programming languages to express more parallelism
  - Improve debugging

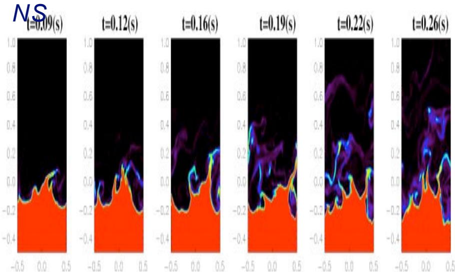
What Did We Do:

- Standardized on a single Open Source development environment: Linux
- Created Open Source scalable libraries and performance tools
- Designed scalable, parallel I/O semantics
- Developed sophisticated, multi-component application frameworks
- Made programming environments **more** rich and complex, adding perl, python, and dynamically loaded libraries
- Addressed power consumption and density
- *Refactored system architecture and software*
- Complained about debugging

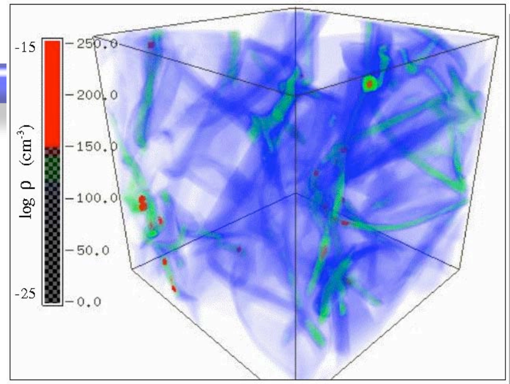
The Petaflop Applications Have Already Been Written



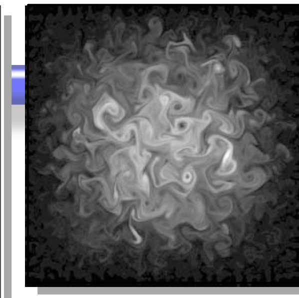
Shortly: Relativistic accretion onto NS



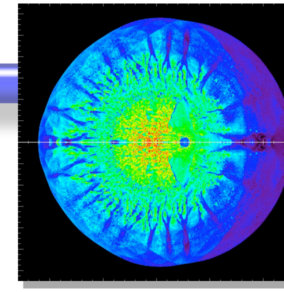
Wave breaking on white dwarfs



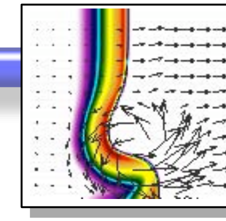
Gravitational collapse/Jeans instability



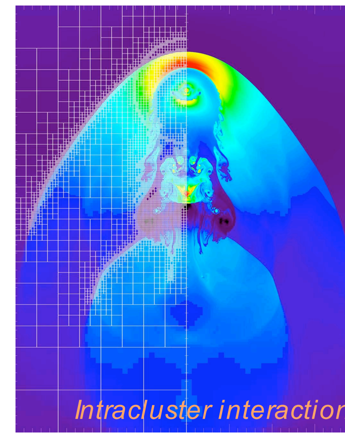
Compressed turbulence



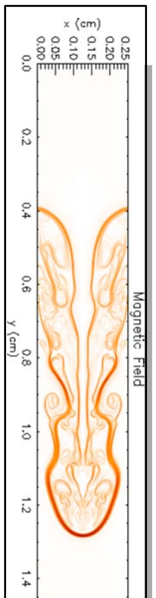
Type Ia Supernova



Flame-vortex interactions

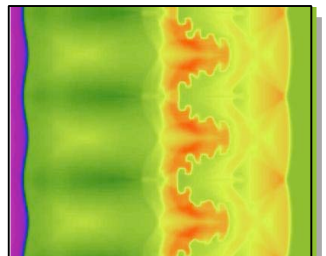


Intracluster interactions

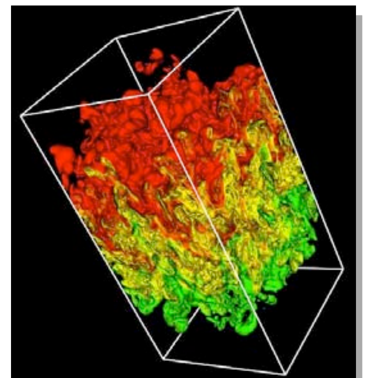


Magnetic Rayleigh-Taylor

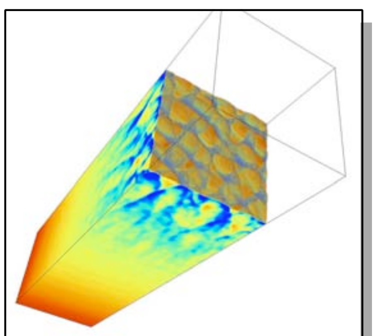
Nova outbursts on white dwarfs



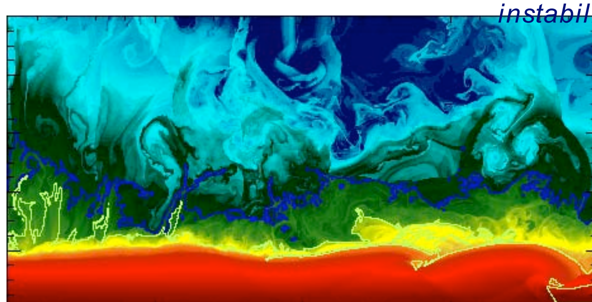
Laser-driven shock instabilities



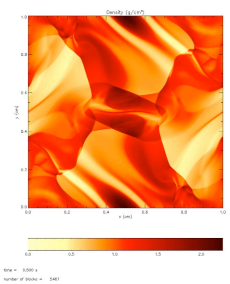
Rayleigh-Taylor instability



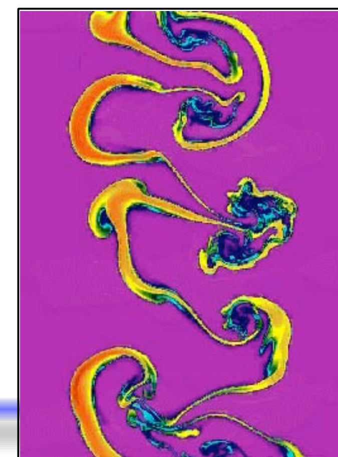
Cellular detonation



Helium burning on neutron stars



Orzag/Tang MHD vortex



Richtmyer-Meshkov instability

We Have Two Challenges

- Scale existing applications on BG/L. Some will be petaflop candidates
  - Improve floating point code
  - Replace bottlenecks with scalable I/O, data distributions, and algorithms
- Develop a reusable architecture and Open Source system software
  - The community is too small for 3 different system software models
  - The community must adopt and extend the basic architecture

The Context for System Software: The Evolving Architecture

- Large Flat Clusters
  - Berkeley NOW makes Top500, Solaris, 1997
- Basic architecture evolves via Open Source and Linux -- system software scales poorly
- Hierarchical Linux clusters evolve
  - Cplant, Chiba City, etc
  - No adoption as target for system software devl.
- Clusters of SMPs and point solutions let systems reach 1000s of CPUs (with difficulty)
- IBM, Cray, and others develop highly scalable hierarchical platforms combining Linux, proprietary components, and Open Source HPC components
- **>>You are Here<<**
- Community embraces new platforms and develops Open Source components and extends system software capabilities

This New Model

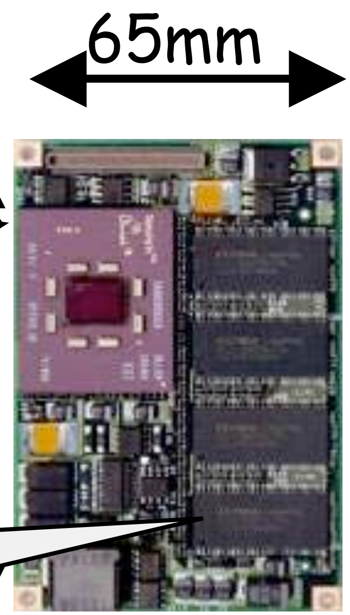
- "What OS does BG/L run?"
  - Service Node: Linux SuSE SLES 8
  - Front End Nodes: Linux SuSE SLES 9
  - I/O Nodes: IBM-created Embedded Linux
  - Compute Nodes: Home-brew OS
- "What OS does Red Storm run?"
  - Service Nodes: Linux
  - RAS Nodes: Linux
  - I/O Nodes: Linux
  - Compute Nodes: Home-brew OS
- Extremely large systems run an "OS Suite"
  - **Functional Decomposition** trend lends itself toward a customized, optimized point-solution OS
  - **Hierarchical Organization** requires software to manage topology, call forwarding, and collective operations

JST-CREST “Megascale” Project

- V1: TM5900
- 0.9GFlops@7W
- ⇒ 112.5MFlops/W
- L2C=512KB

- V2: Effecion TM8800
- 2.4GFlops@5W
- ⇒ 480MFlops/W
- 512MB DDR-SDRAM

- 256MB SDRAM
  - (512M DDR in V2)
- 512KB flash

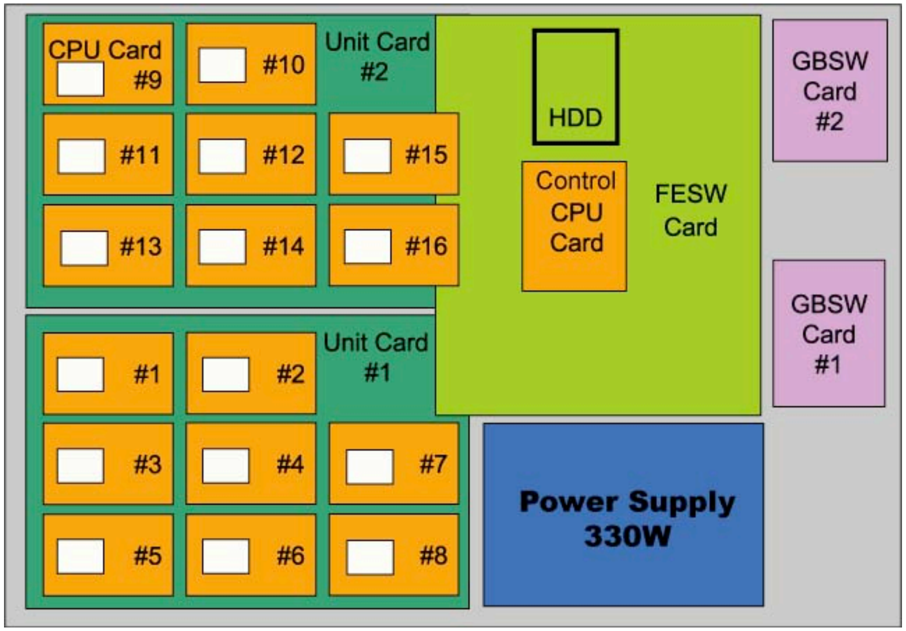


- PI: Hiroshi Nakashima (Toyohashi IT) – Megascale Cluster Federation (Grid) programming
- Co-Pis
  - Hiroshi Nakamura (U-Tokyo) – Low power processor architecture
  - Mitsuhisa Sato (Univ. Tsukuba) – Low power compiler and runtime
  - Taisuke Boku (Univ. Tsukuba) – Dependable multi-way interconnect
  - Satoshi Matsuoka (Titech) – Dependable and Autonomous Cluster Middleware

Courtesy: Satoshi Matsuoka

MegaProto Packaging (1U Chassis)

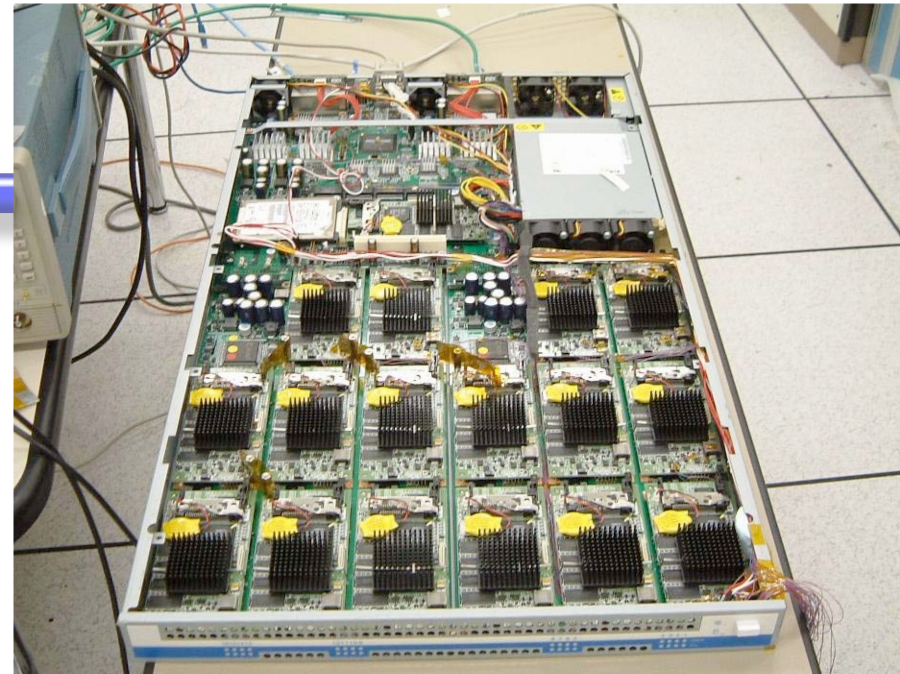
- 16+1 processor cards (fanless)
- 2 types of built-in networks on MB
  - Dual GbE, 16 x 2 (Internal) + 8 x 2(external) ports, internally switched
  - Control Network (100Base-T)
- Processor Card Upgradable (PCI-X based interface)
- Control CPU, Local HDD
- 330W Power Supply

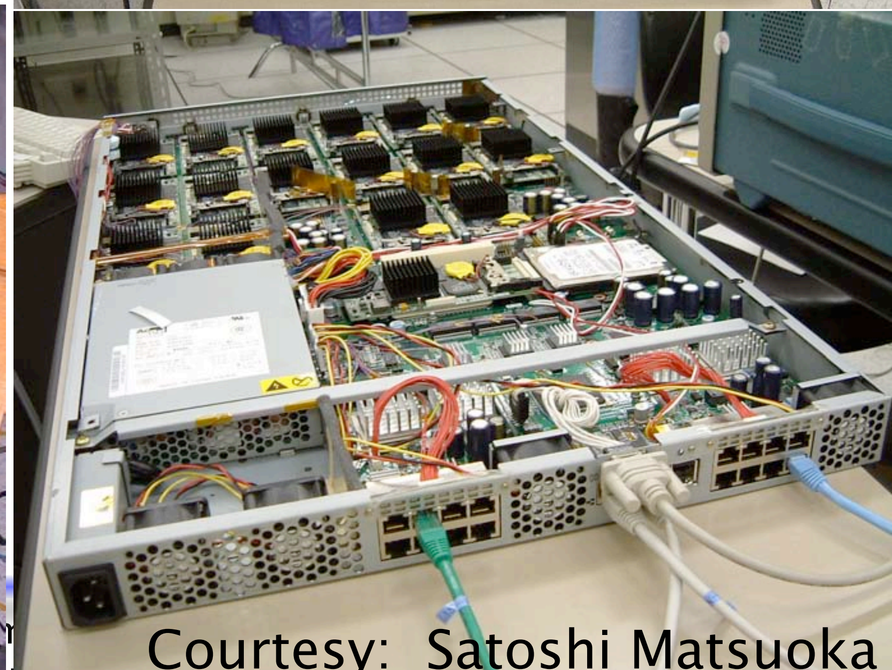


Courtesy: Satoshi Matsuoka

The Systems

- Fabrication by IBM Japan (same fab team as BlueGene/L)
- Initial 2 Units (32 PE) delivered, being tested
- 32 proc Linux Cluster





Courtesy: Satoshi Matsuoka

Our Challenge:
Optimize The SW Stack For New Architecture Class

Science and Engineering Applications

Runtime and User Environments	Data Management and Visualization
F90/C/C++/MPI/ Libraries	Performance Management/Analysis
HW Abstraction/OS/Linux	Systems Management/Configuration

Scalable Hardware Platforms

Including:

- Parallel File Systems
- High Performance MPI-2 & MPI-IO
- Global Arrays, UPC, CAF
- Parallel workflow and scripting
- Job and resource management
- Pipes to real-time viz
- Fault tolerance
- Collective operating system calls
- Performance tools that work cooperatively across all components
- Improve debugging

Good News: Progress Already!

- ANL has developed a Linux I/O node toolkit that can be distributed to developers
  - Special thanks to LLNL & IBM
- What is done:
  - Linux kernel, config., compile & ramdisk tools, etc.
- How it is being used:
  - Extend capabilities of I/O node
  - Build kernel modules
  - Build performance tools (TAU extension)
- The Open Source model is working
  - The World's First Parallel Open Source File System for BG/L is now working!
- We will release the env. after we finish skiing
- More utils coming…

Goals for Workshop

- Learn about the system software currently available and in development for BG/L
- Share experiences and solutions
- Prioritize system software enhancements and requirements
- Find areas where community Open Source development can extend the BG/L environment
- Organize community efforts and build collaboration

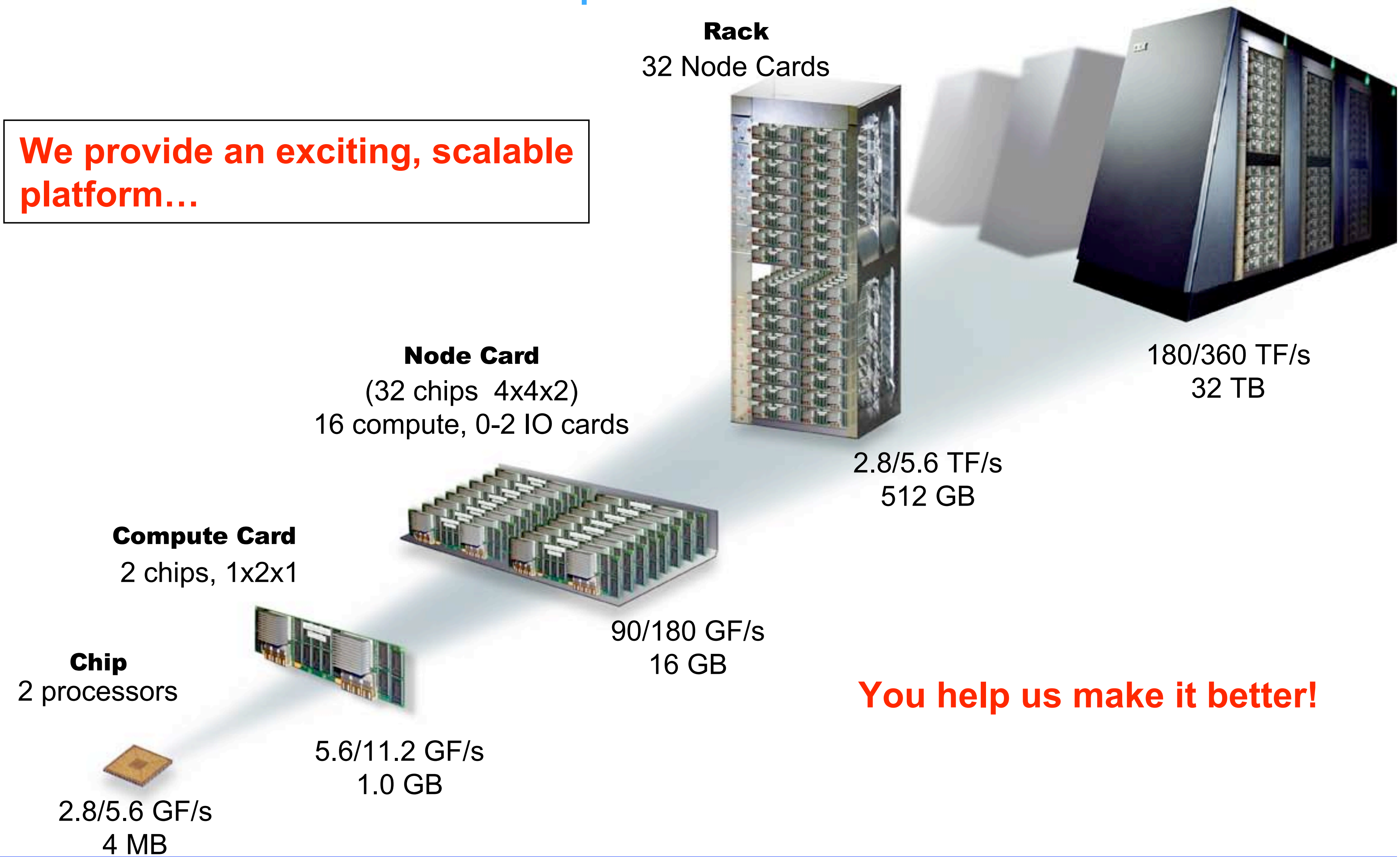
Blue Gene System Software: Let's collaborate!

Manish Gupta

IBM Thomas J. Watson Research Center

Blue Gene Partnership

We provide an exciting, scalable platform…



You help us make it better!

Blue Gene Partnership Goals

- Push system scalability to unprecedented levels
- Support high productivity – make system easier to use and manage
- Make system useful for a broader class of applications

Blue Gene Partnership Goals

- Push system scalability to unprecedented levels
- Support high productivity – make system easier to use and manage
- Make system useful for a broader class of applications

- Impact on Blue Gene/L product
  - Constraints – supporting changes to base software
  - Opportunities – many areas to augment IBM offering

Blue Gene Partnership Goals

- Push system scalability to unprecedented levels
- Support high productivity – make system easier to use and manage
- Make system useful for a broader class of applications

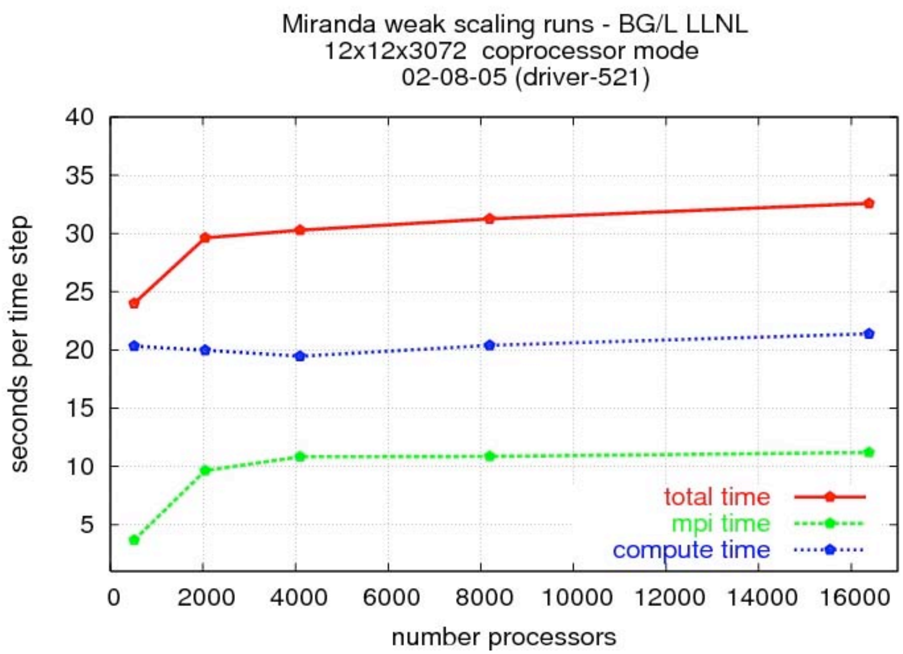
- Impact on Blue Gene/L product
  - Constraints – supporting changes to base software
  - Opportunities – many areas to augment IBM offering

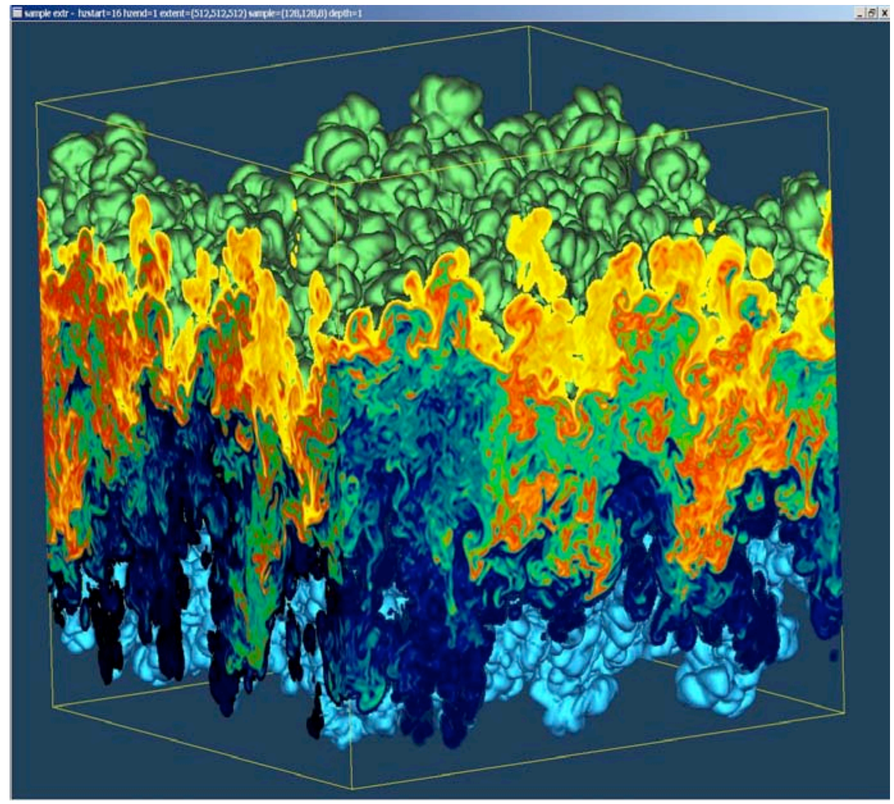
- Impact on Blue Gene/P design

Status Summary

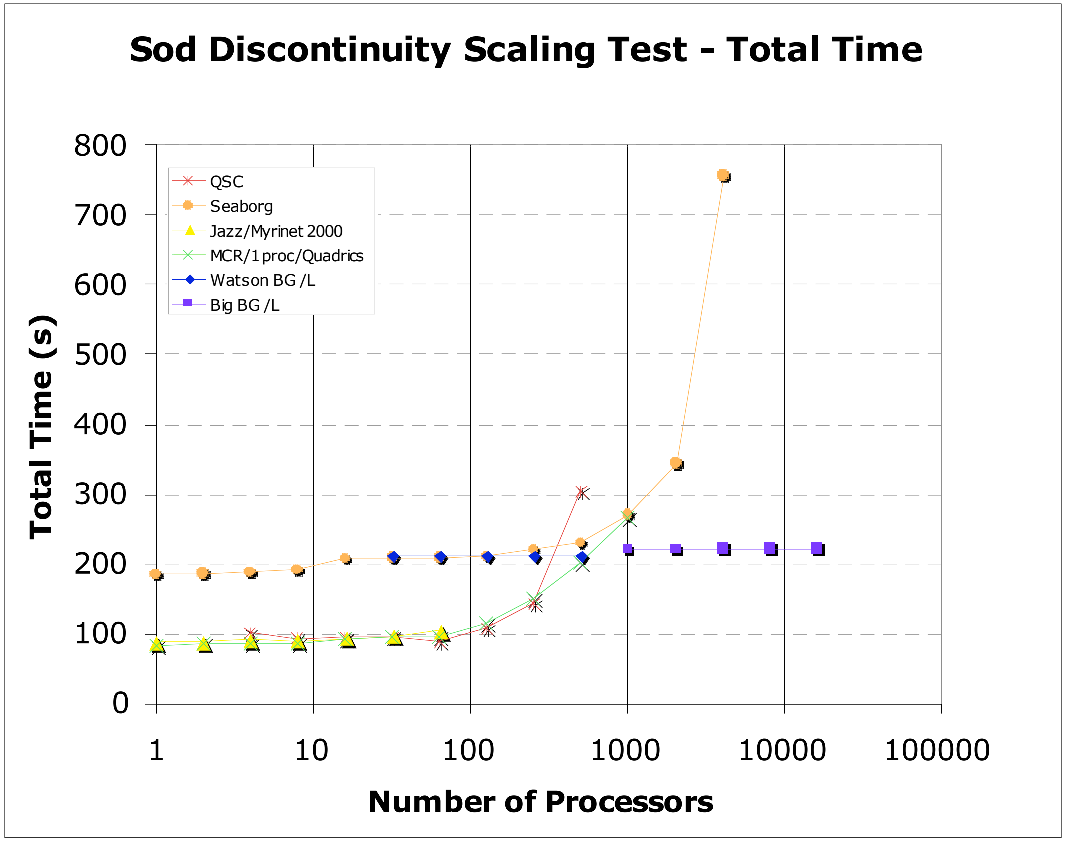
- 16 racks (16,384 nodes, 32768 processors) at Rochester and LLNL
  - Another 16 racks on LLNL floor
- 70.72 TF/s sustained Linpack
  - #1 on TOP500 list
- Various applications and benchmarks executed – IBM and LLNL
  - Highest ever delivered performance on many applications

Miranda Weak Scaling on BG/L





FLASH: Astrophysics Code from Argonne National Lab
SCALING TO 16x1024 nodes on Blue Gene/L



Big BGL: 16 Racks, coprocessor, 440

Jazz : 350node, 2.4GHz Xeon, ANL

MCR : 1152node,2.4GHz Xeon,LLNL

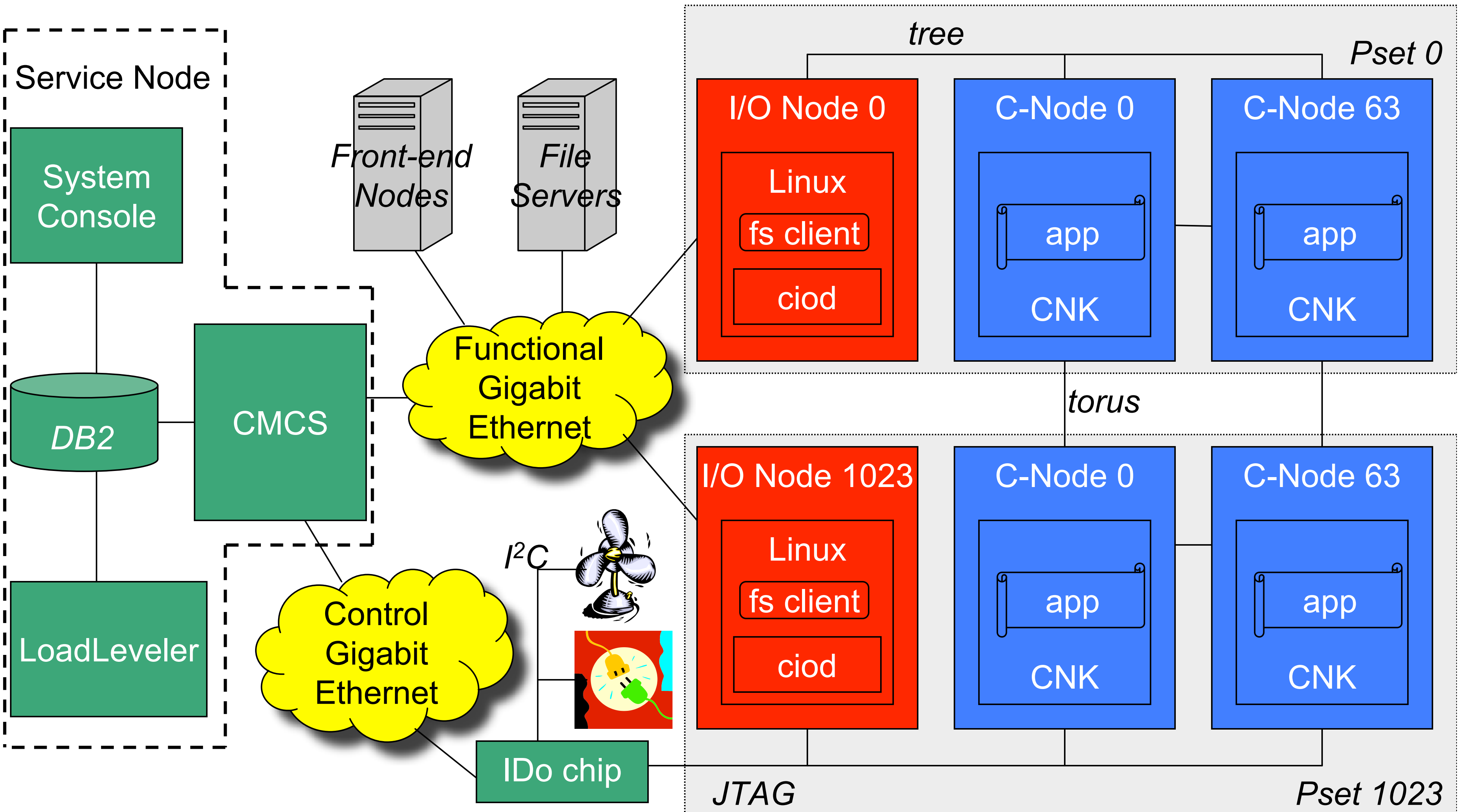
Seaborg: IBM SP, 1.5GF/node NERSC

QSC: 256nodex4way HP Alpha, LLNL

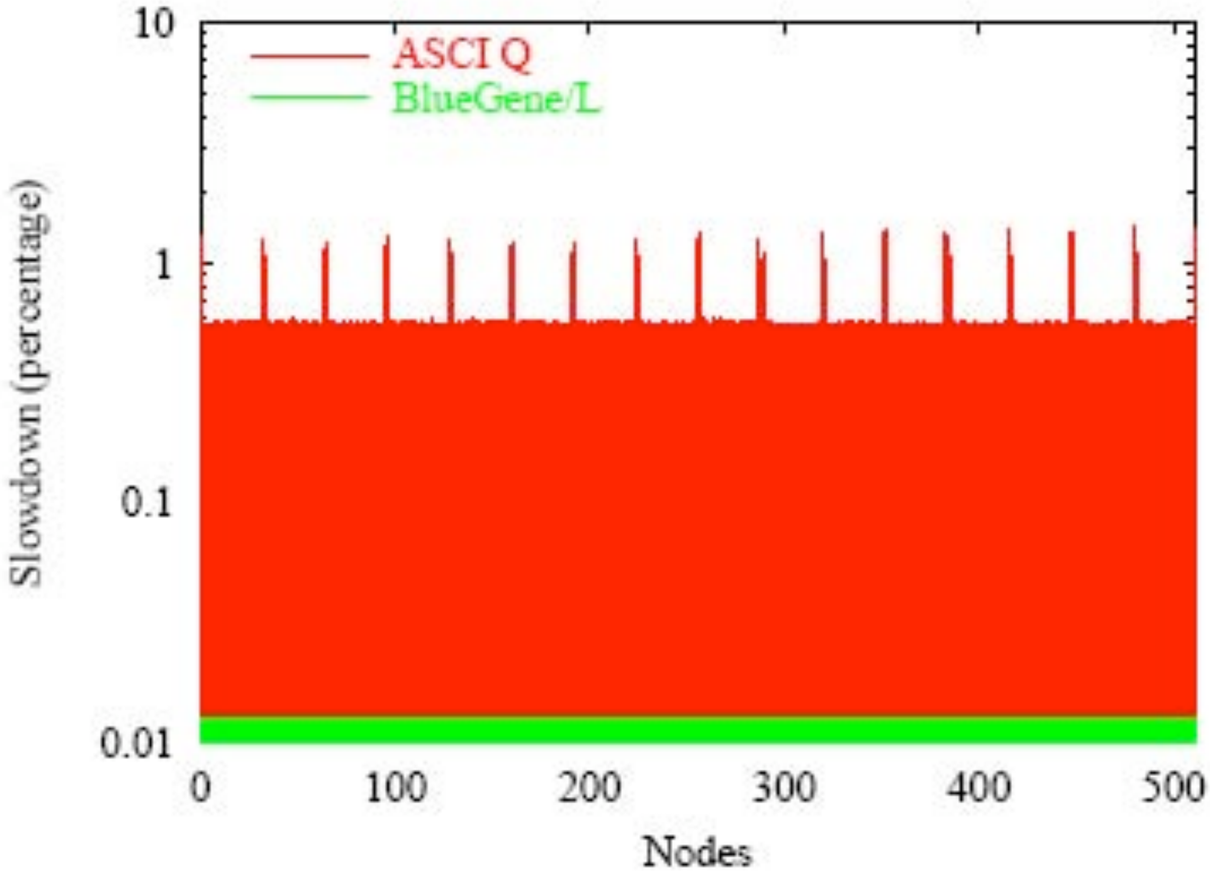
Much work in progress…

- Parallel file system (GPFS) under installation and test
- Job scheduling solution (LoadLeveler) coming soon
- System management enhancements
- MPI enhancements
- Math libraries (full ESSL, MASS, MASSV) being developed
- Performance tools being developed
- Compiler enhancements

BlueGene/L System Architecture

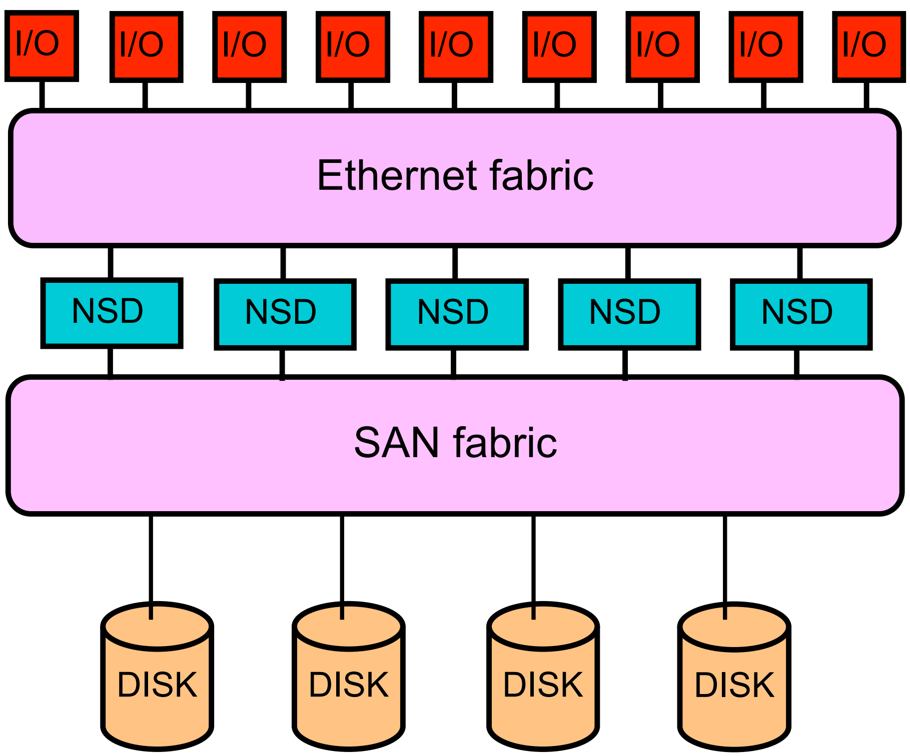


Noise measurements (from Adolphy Hoisie)



Ref: Blue Gene: A Performance and Scalability Report at the 512-Processor Milestone, PAL/LANL, LA-UR- 04-1114, March 2004.

Parallel File System for BlueGene/L (GPFS)

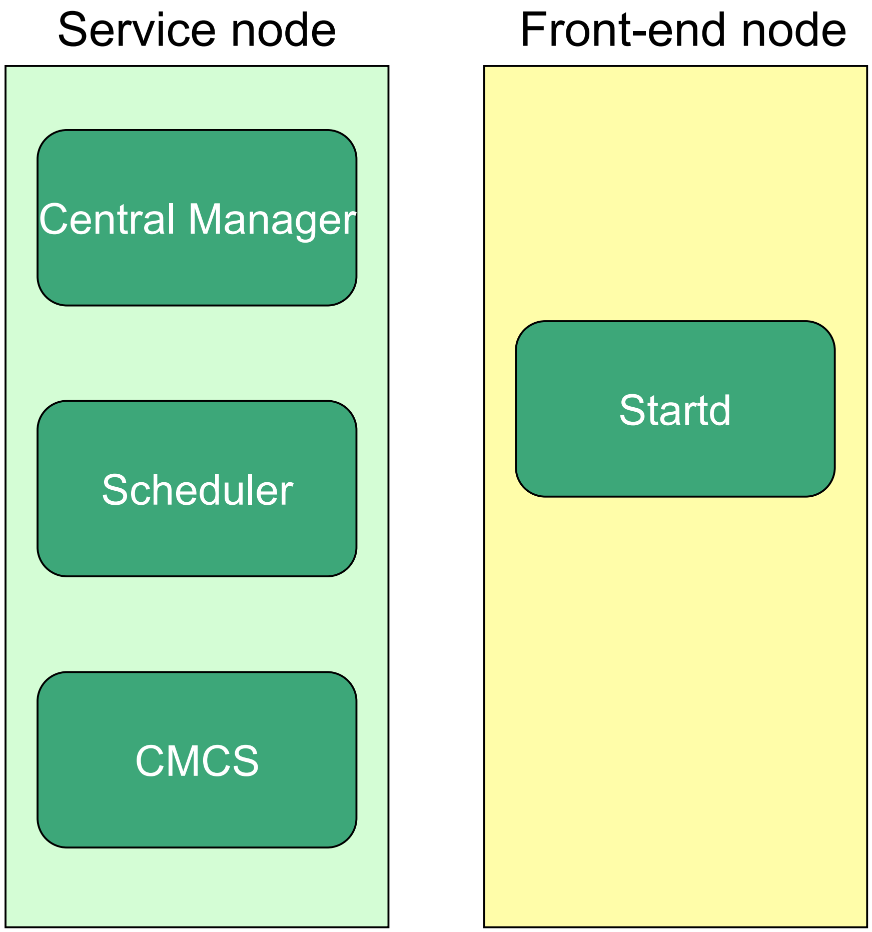


- GPFS solution for BlueGene/L is 3-tiered
  - First tier consists of the I/O nodes, which are GPFS clients – currently run NFS clients
  - Second tier is a cluster of NSD (Network Shared Disk) servers
  - Third tier is a set of storage devices, typically fiber channel or iSCSI
- First-to-second tier interconnect has to be Ethernet
- Second-to-third tier can be fiber channel loop, fiber channel switch, or Ethernet (for iSCSI)
- Choice of NSD servers, SAN fabric and storage devices depends on specific requirements

Job Scheduling in BlueGene/L

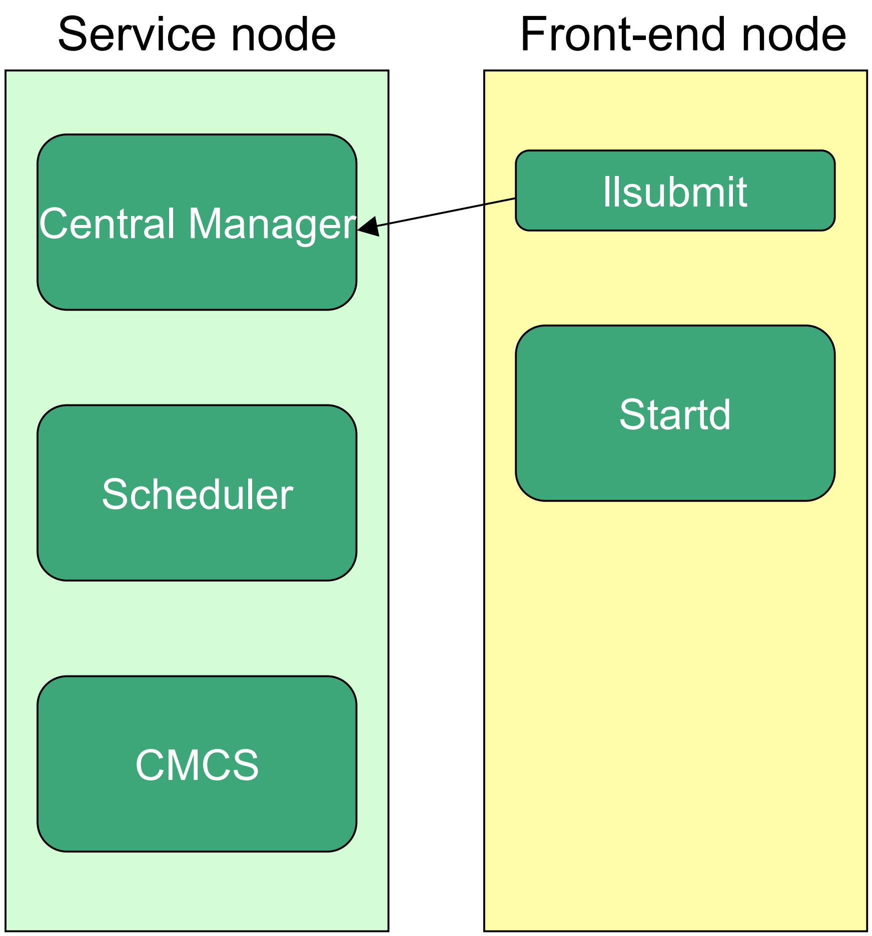
- LoadLeveler solution
  - BG/L specific job scheduler plugged into LoadLeveler as external scheduler
  - Working on a integrated, internal scheduler, solution
- Job scheduling strategies can significantly impact the utilization of large computer systems
  - Machines with toroidal topology (as opposed to all-to-all switch) are particularly sensitive to job scheduling – this was demonstrated at LLNL with gang scheduling on Cray T3D
  - BG/L scheduling strategies leveraging BG/L unique topology features can significantly enhance system utilization – from 45% to almost 90% (depends on workload)

LoadLeveler for BlueGene/L



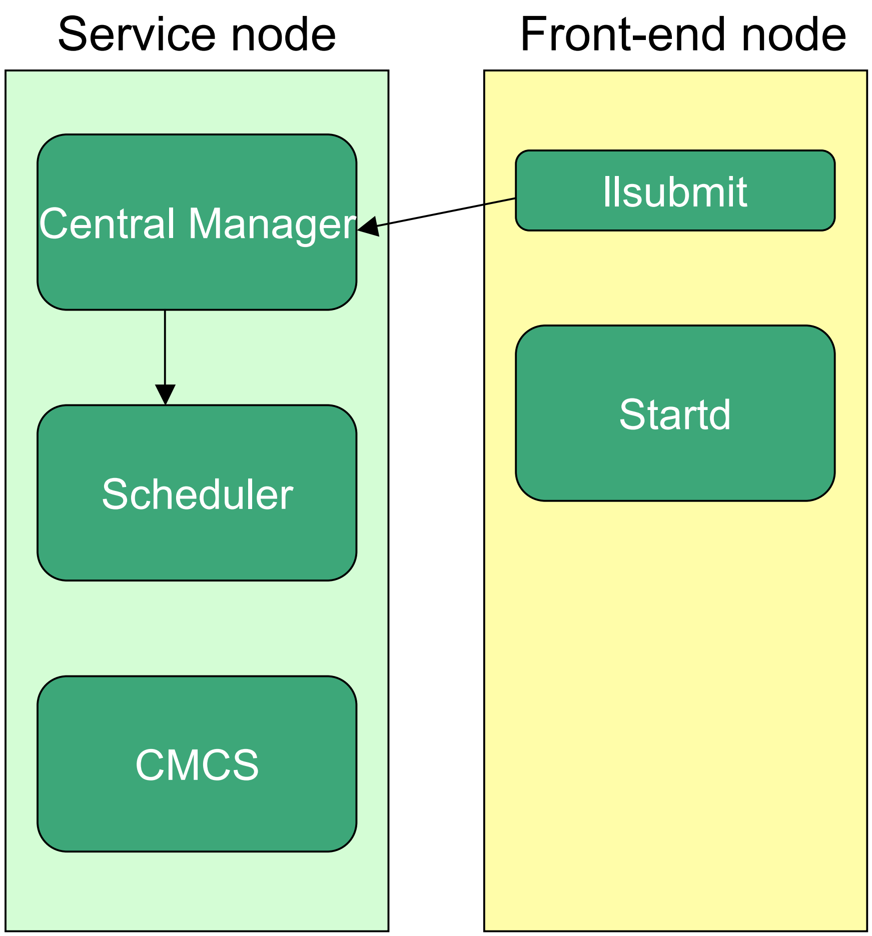
- The BlueGene/L implementation of LoadLeveler is contained entirely in the service and front-end nodes
- The service node runs the *Central Manager* daemon and external scheduler
- Front-end nodes run the *Startd* daemon

LoadLeveler for BlueGene/L



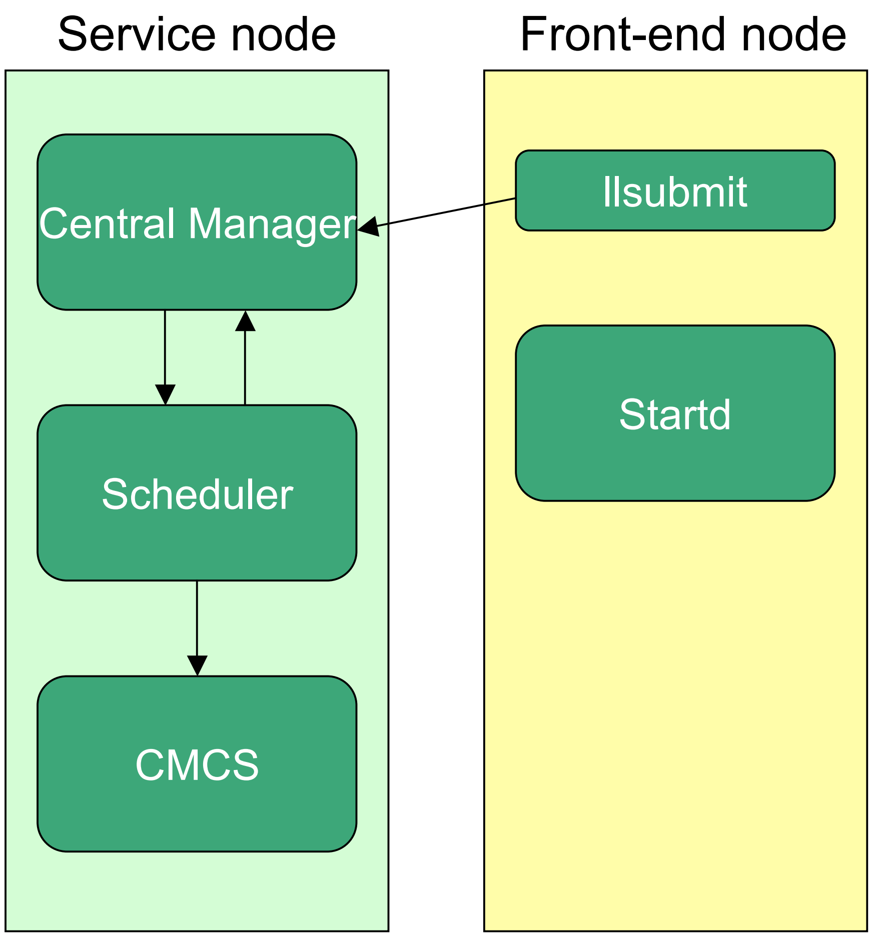
- The user submits a job from the front-end node
- The llsubmit command contacts the Central Manager to enqueue the job for executions

LoadLeveler for BlueGene/L



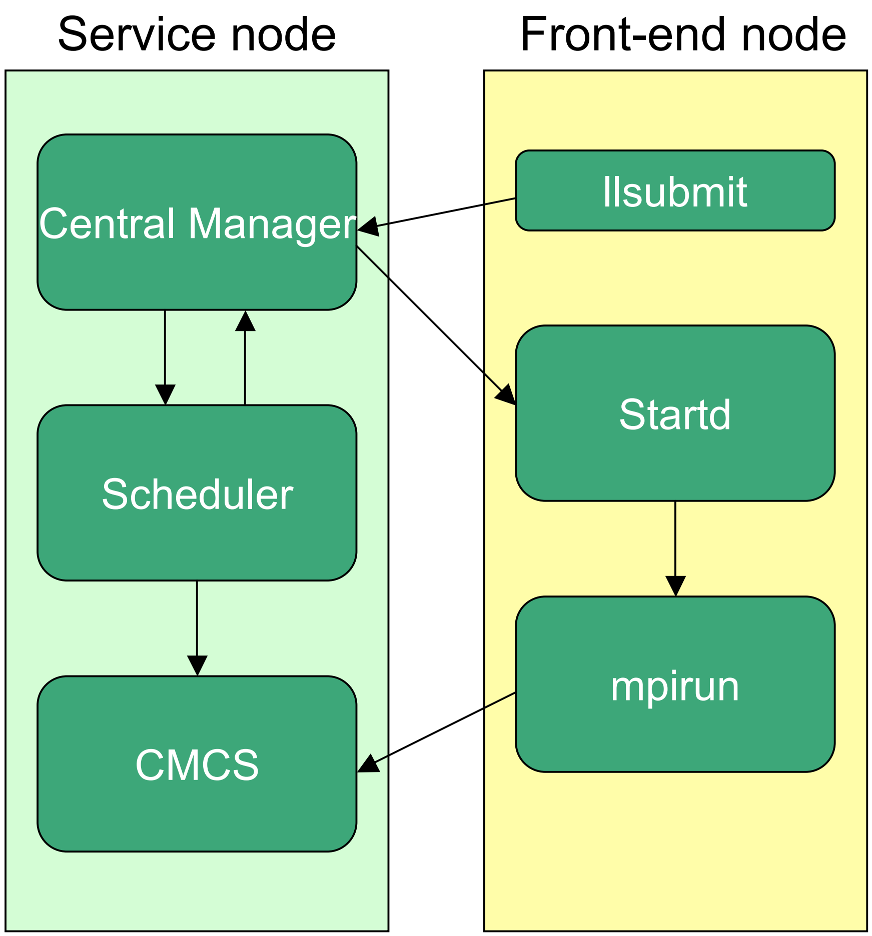
- The user submits a job from the front-end node
- The llsubmit command contacts the Central Manager to enqueue the job for executions
- The scheduler retrieves the queue of jobs to execute and makes policies decisions

LoadLeveler for BlueGene/L



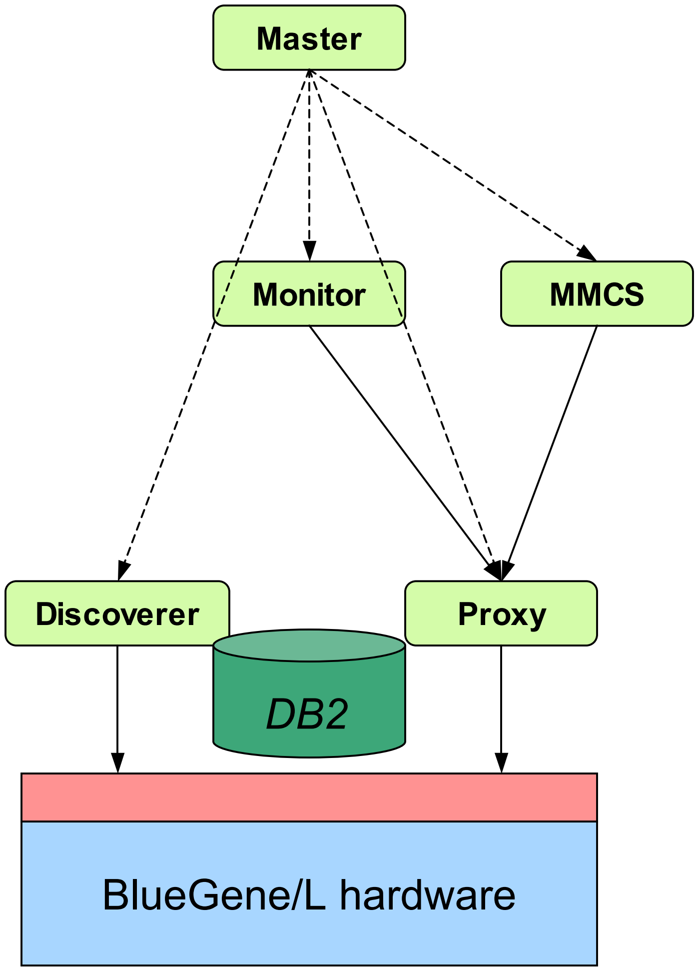
- The user submits a job from the front-end node
- The llsubmit command contacts the Central Manager to enqueue the job for executions
- The scheduler retrieves the queue of jobs to execute and makes policies decisions
- The scheduler uses control system services to create a machine partition and instructs the Central Manager to start the job

LoadLeveler for BlueGene/L



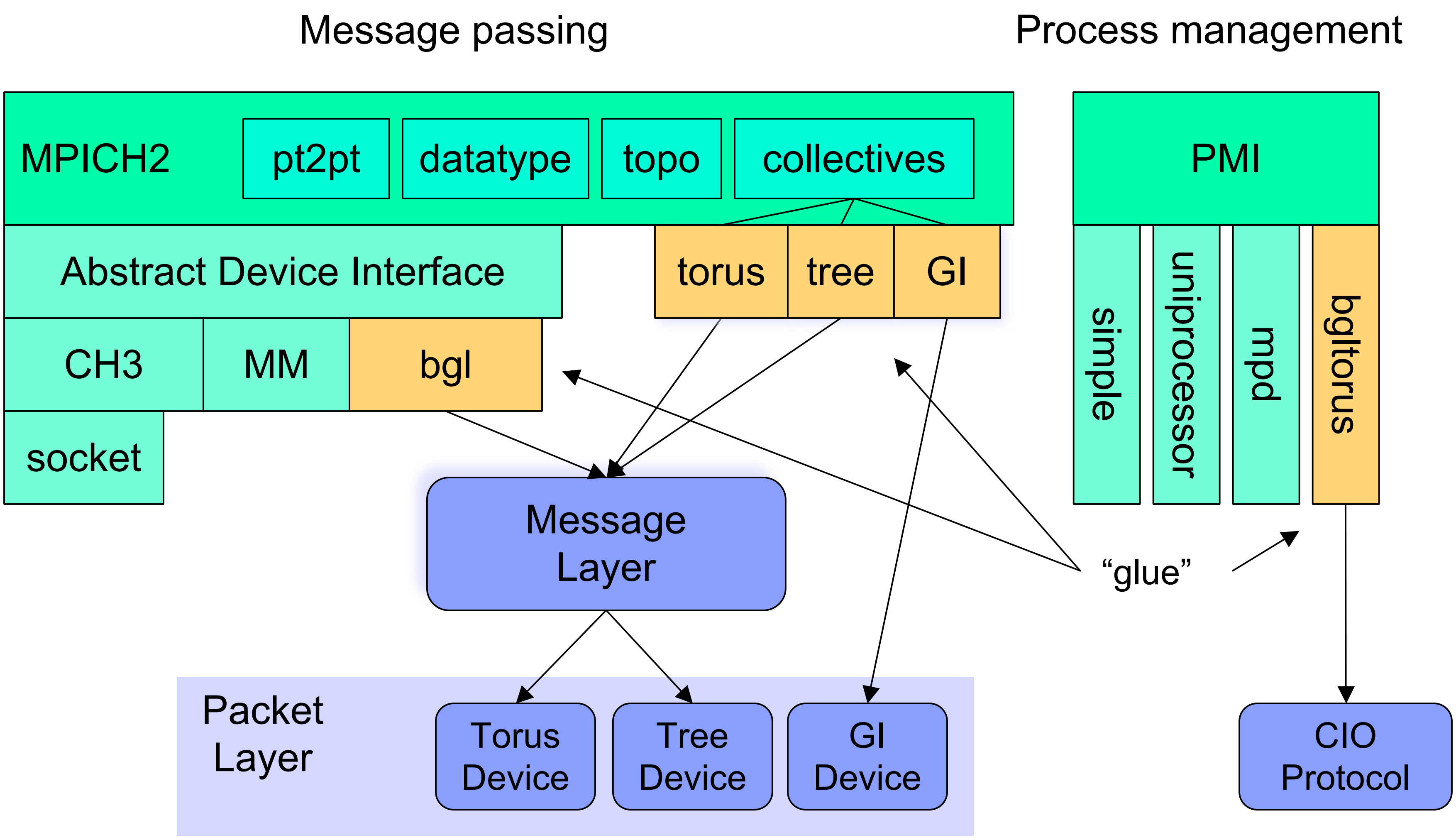
- The Central Manager contacts the Startd daemon on the front-end node to launch mpirun
- The mpirun process uses control system services to launch the actual application processes in the partition created by the scheduler
- The mpirun process stays in the front-end node as a proxy of the user application
- Debuggers (e.g., TotalView) work by attaching to the mpirun process

Control System – Components



- Master creates, monitors, and restarts the other processes
- Discoverer finds and initializes new hardware
- Proxy virtualizes the IDo hardware, providing reliable and atomic connection
- Monitor monitors environmentals, such as temperature and voltages
- MMCS configures and IPLs partitions of the machine, bringing those partitions to a user-architected state

MPI – based on MPICH2 from ANL



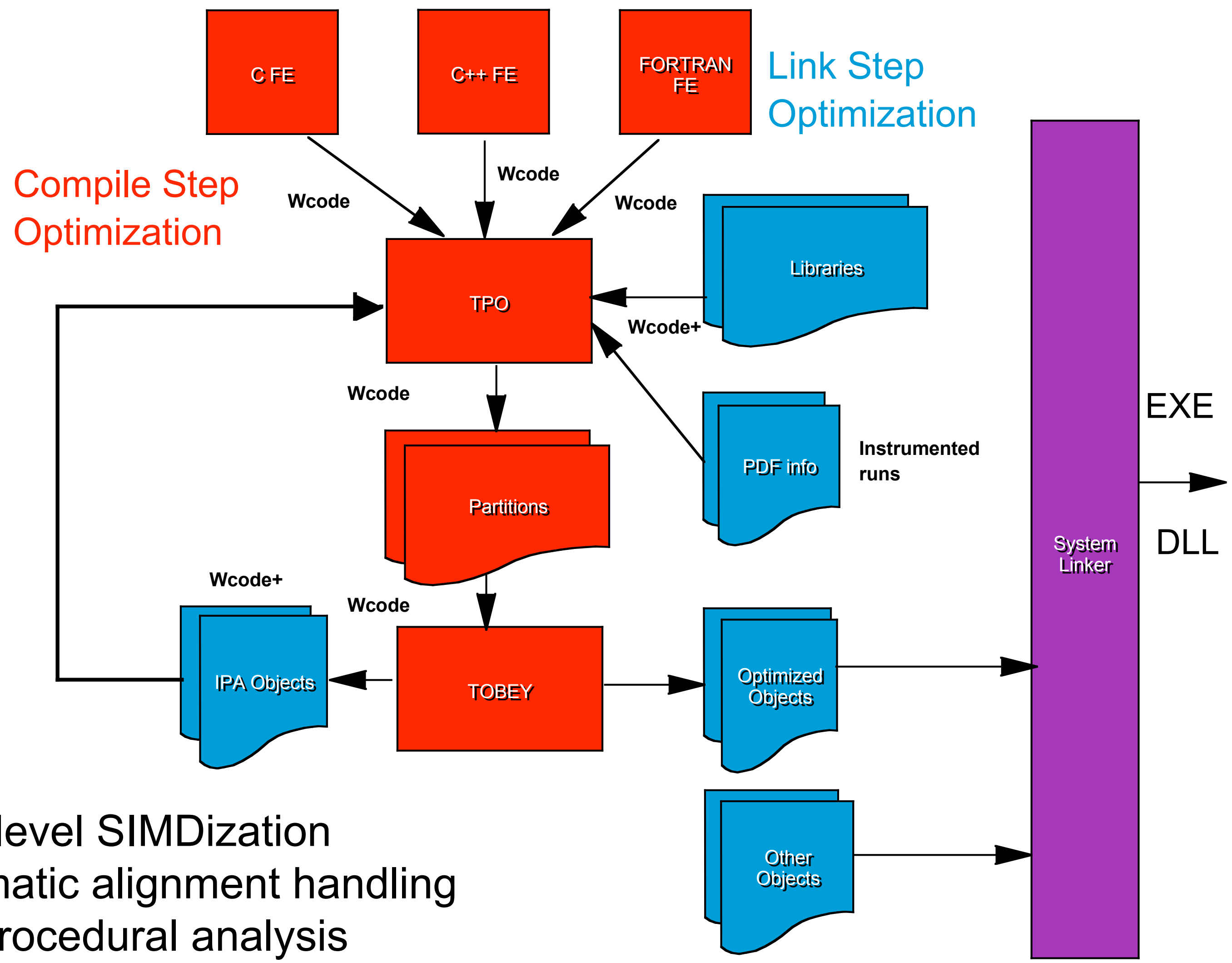
MPI enhancements

- Higher levels of scalability
  - Continued enhancements of collectives
  - Adaptive buffer management with flow control
  - Support for interrupts
  - Adaptive protocol selection with compiler analysis
- MPI-IO support
  - BG/L specific optimizations
  - Optimize GPFS based on higher level view

Strategy to Exploit SIMD FPU

- Automatic code generation by compiler (-qarch=440d)
  - Single FPU fallback: -qarch=440
- User can help the compiler via pragmas and intrinsics
  - Pragma for data alignment: *__alignx(16, var)*
  - Pragma for parallelism
    - Disjoint: *#pragma disjoint (*a, *b)*
    - Independent: *#pragma ibm independent* loop
  - Intrinsics
    - Intrinsic function defined for each parallel floating point operation
      - E.g.: *D = __fpmadd(B, C, A) => fpmadd rD, rA, rC, rB*
    - Control over instruction selection, compiler retains responsibility for register allocation and scheduling
- Using library routines where available
  - Dense matrix BLAS – e.g., DGEMM, DGEMV, DAXPY
  - FFT
  - MASS, MASSV

IBM Compiler Architecture

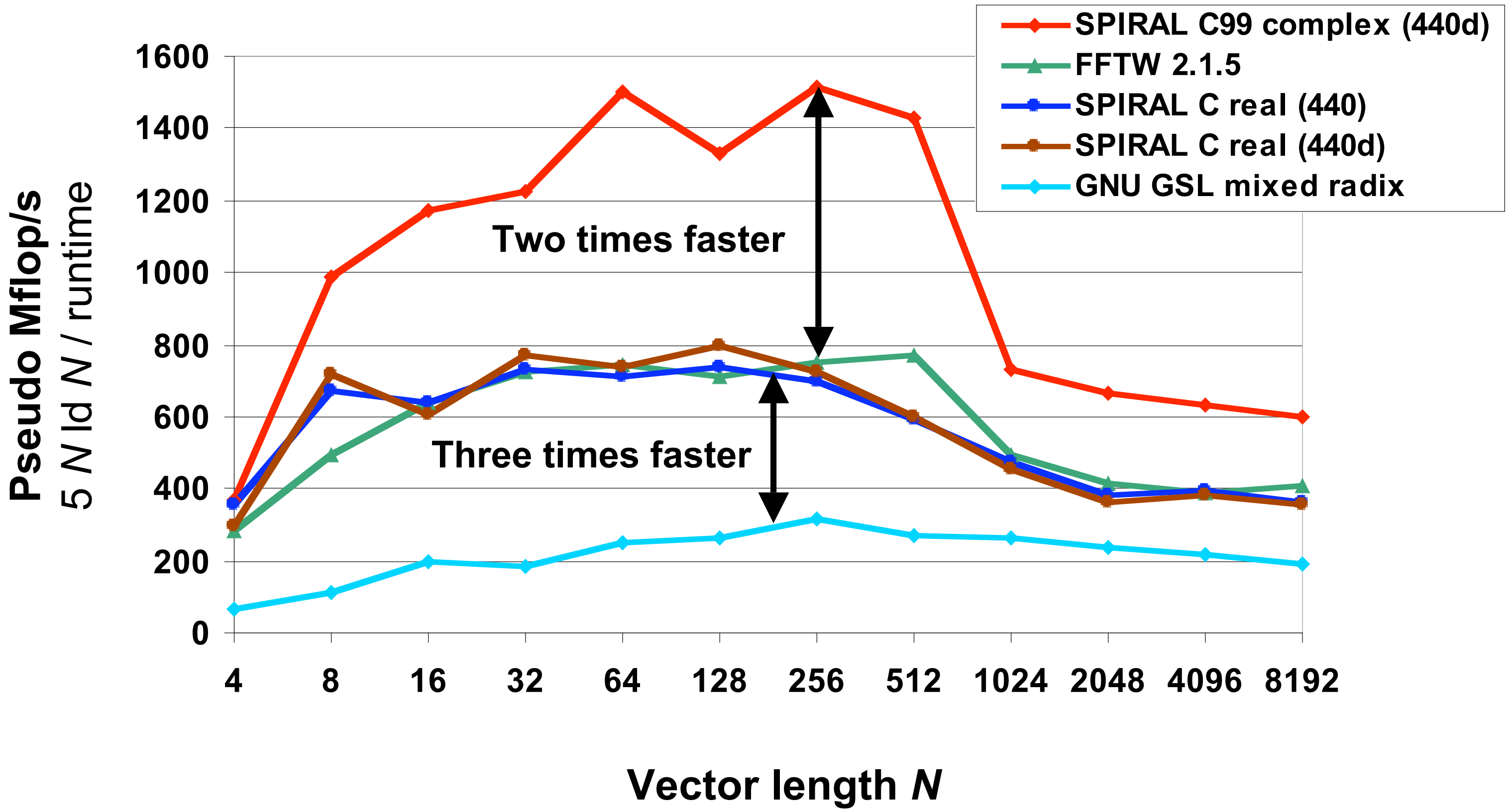


Loop level SIMDization
Automatic alignment handling
Interprocedural analysis

Math Libraries: ESSL

- Started with small subset (of ~500 routines)
  - Mainly dense matrix kernels – DGEMM, DGEMV, DDOT, DAXPY etc.
- Using ESSL source code to drive compiler testing and exploration of complete ESSL support
  - Status: Nearly complete functionality available using –O3 –qarch=440
  - Currently investigating SIMD FPU issues, performance enhancements
  - Expected general availability – Nov 2005
- FFT
  - Technical University of Vienna developing FFT library optimized for BlueGene/L – effective use of the SIMD FPU

FFT Measured Performance



DFT 2^n, complex, double precision

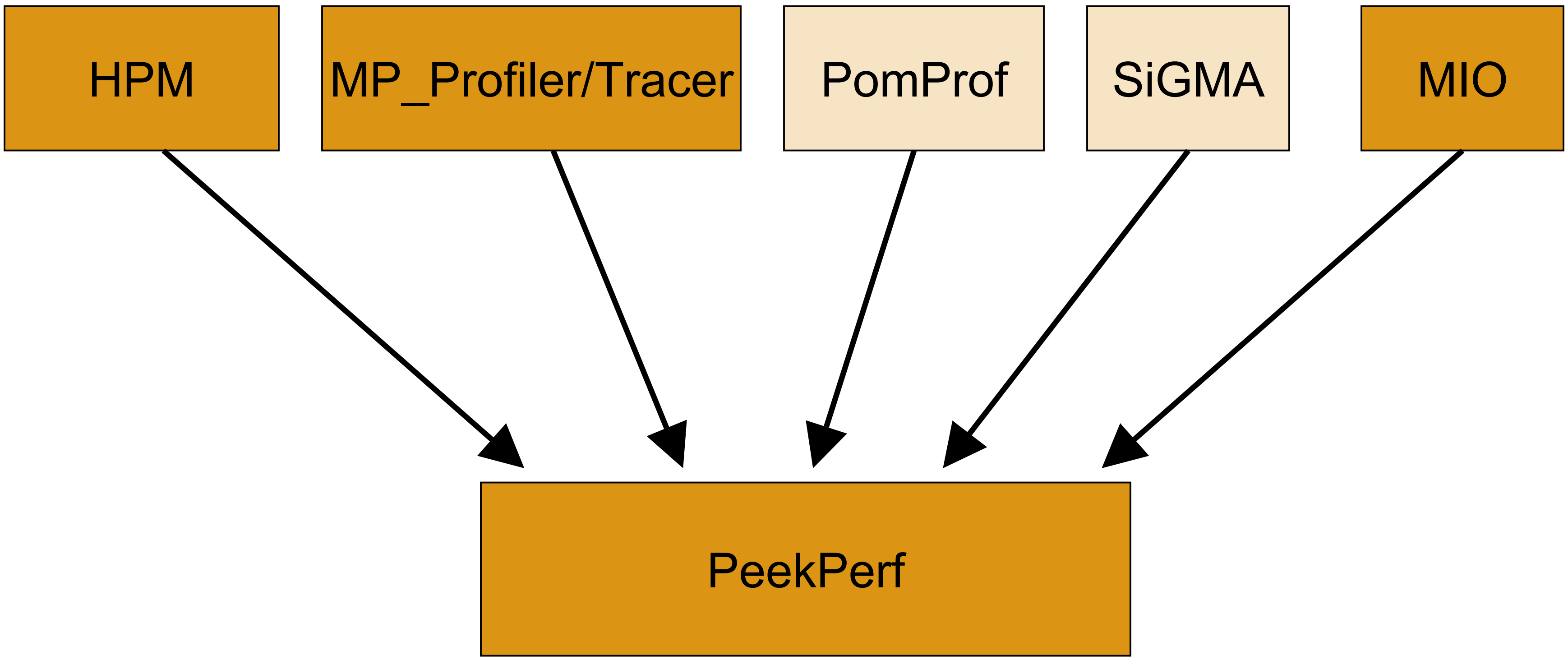
VisualAge XL C 7.0 for BlueGene/L options: -O3 qnostrict – qarch=440/440d

BlueGene/L DD2 prototype at IBM T.J. Watson Research Center
Single BlueGene/L CPU at 700 MHz (one Double FPU)

Math Libraries: MASS and MASSV

- Math intrinsic routines – e.g., square root, exponential, sine, cosine (~50 routines)
  - Traditionally supported on pSeries platforms with hand-tuned assembler routines
  - Up to factor of 5-20x performance boost over naïve versions
- BG/L: Novel approach using special compilation of versions written in C
  - Being deployed by Toronto compiler team on Apple platform
  - Complete set of routines available using this approach
  - Reciprocal, square root, reciprocal square root, exponential, logarithm, cube root optimized for BG/L – prioritized based on early applications
  - Expected availability of MASS, MASSV – June 2005

Performance Tools – based on IBM HPCT



Additional tools - Code profiler (gprof, Xprofiler), Mapping tool for 3D torus topology
New challenge – scalability of tools

Advanced Programming Models

- Global Arrays
  - Prototype implementation of ARMCI (active message library) on BG/L
    - ARMCI used as a driver for active message libraries
      - » Motivated a rewrite of message layer
  - Performance problems in handling Torus interrupts
    - >10000 cycles currently
  - Prototyping new message layer to provide interoperability between MPI and ARMCI

- UPC
  - Pursued as part of PERCS project
    - Extensive work on front end and compiler at Toronto
  - Port of UPC runtime to Blue Gene feasible

- MATLAB-like environment for linear algebra
  - Collaboration with UIUC

Collaborations: Improving Programmer Productivity

- High performance libraries and packages
  - Computation – ScaLAPACK, sparse matrix BLAS, PDE solvers, PETSc, …
  - I/O – parallel netCDF, parallel HDF5 libraries
    - MPI-IO optimizations
- Performance tools
  - Identification of performance bottlenecks
  - Techniques for scalability
- Programming models
  - MPI enhancements – topology awareness, fault tolerance
  - Global address support – Global Arrays, UPC, Co-Array Fortran

Conclusions

- Blue Gene/L represents a new level of performance scalability and density for scientific computing
- Blue Gene/L system software stack with Linux-like personality for applications
  - Custom solution (CNK) on compute nodes for highest performance
  - Linux solution on I/O nodes for flexibility and functionality
  - MPI is the default programming model, others are being investigated
- Encouraging performance results – excellent scaling to 16K nodes
- Great opportunities for collaboration
  - Complement IBM efforts on BG/L
  - Impact BG/P design

BG/L Compute Node Kernel

Mike Mundy
IBM Rochester

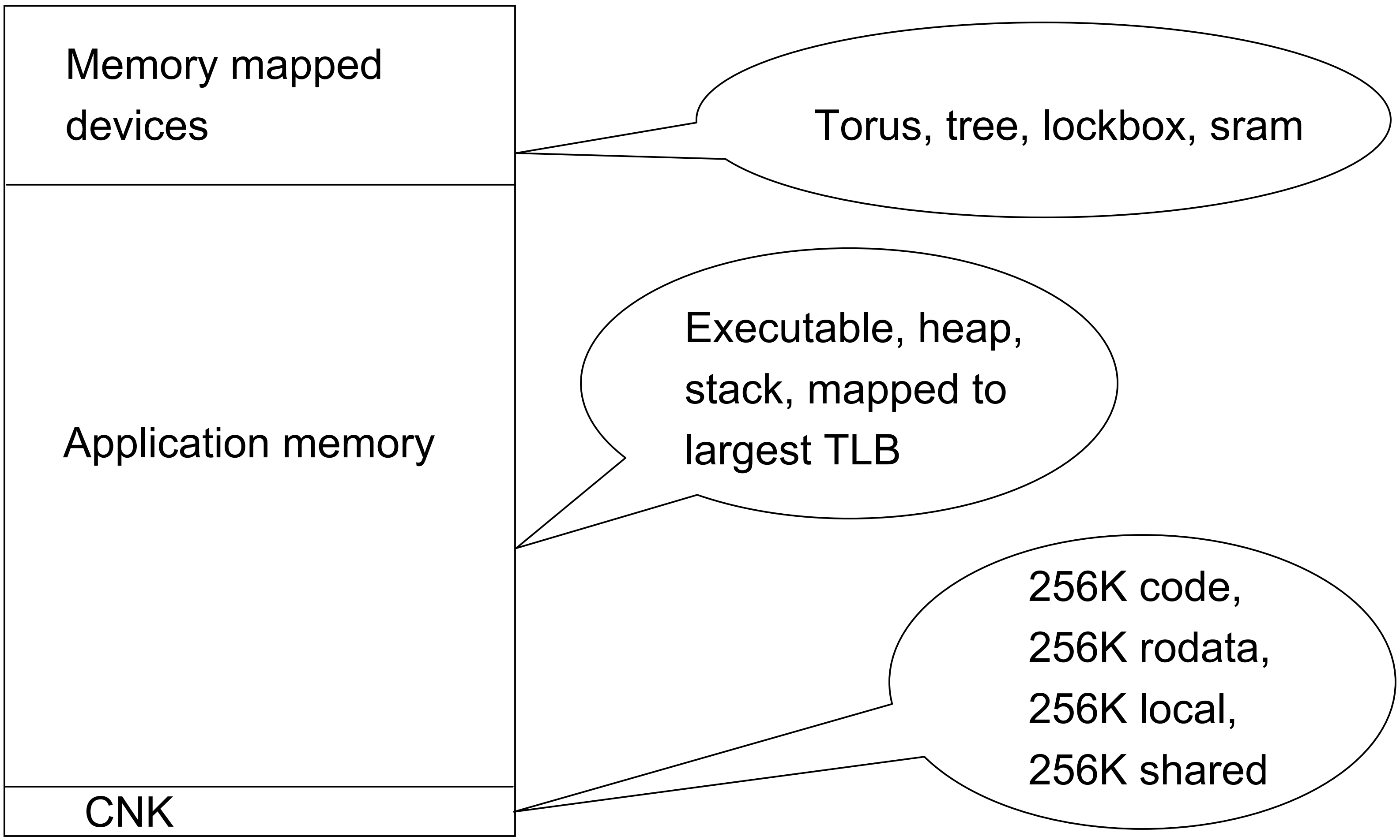
BG/L Compute Node Kernel Agenda

- CNK features and high-level design
- Function shipping to I/O node
- Booting compute nodes and managing jobs

CNK Features

- A simple Linux-like kernel
  - Runs one process at a time
  - Uses small amount of memory – rest for the application
  - Supports attaching debuggers
- CNK provides a subset of the Linux system calls
  - File I/O
  - Directory operations
  - Signals (ANSI C only)
  - Process information
  - Time
  - Sockets
- Goal is to stay out of the way and let the application run

Compute Node Memory Map



CNK Modes

- Coprocessor mode
  - Application runs on processor 0
  - Very limited environment for running code on processor 1
  - MPI uses coprocessor for offloading communications
- Virtual node mode
  - Application is loaded and runs on both processors
  - Memory is divided in half
  - Application is responsible for sharing resources
- Mode is selected at boot time

CNK Limitations

- No support for asynchronous signals using sigaction()
- No support for Linux interprocess communication
- No support for server-side sockets APIs
- No support for poll() or select()
- Limited support for timers

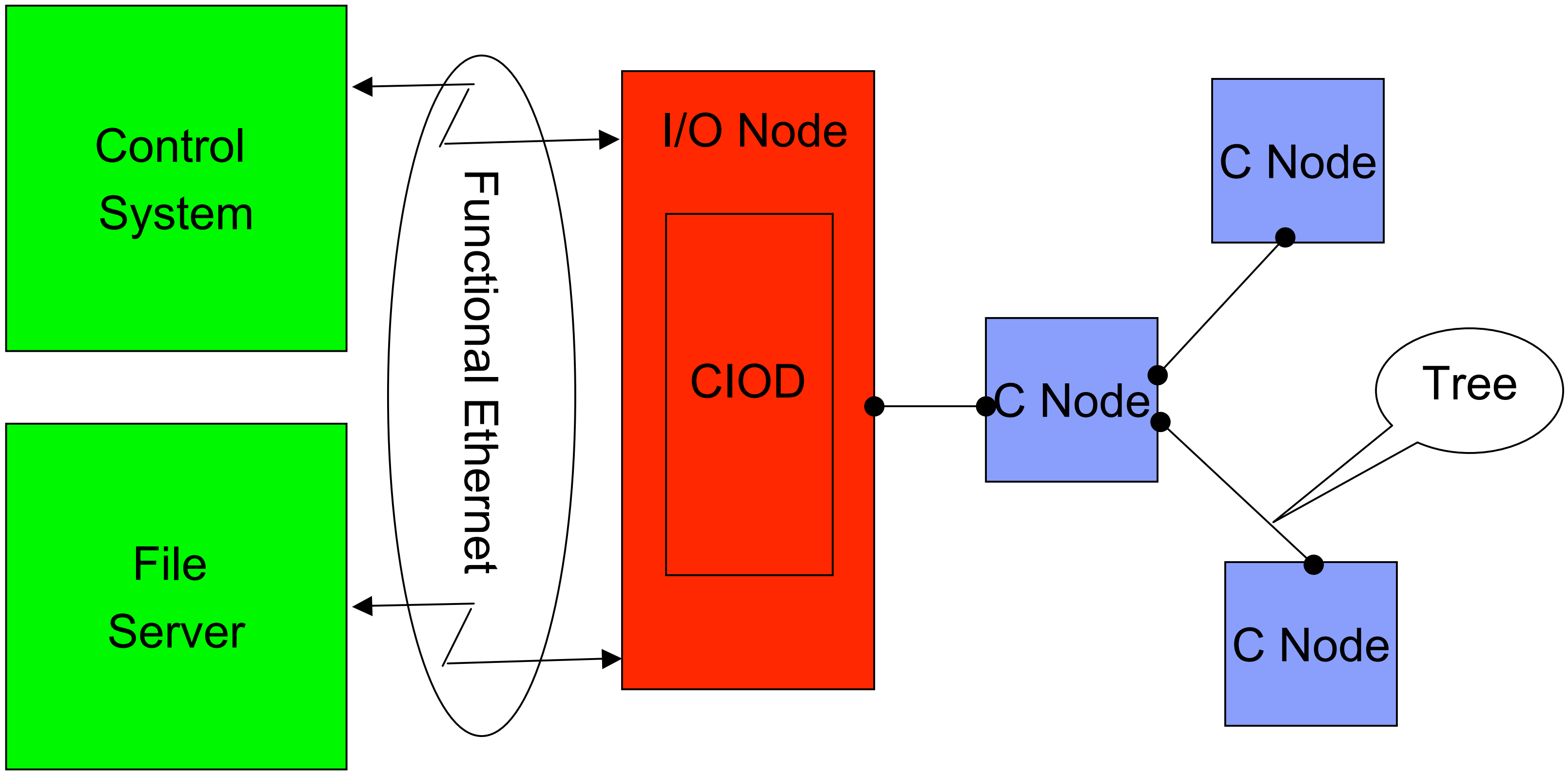
CNK Function Shipping

- All I/O must be processed on the I/O node
- CIOD is a user application running on the I/O node that:
  - Manages the compute nodes for the control system
  - Manages descriptors, working directory, umask for compute nodes
  - Performs all I/O for all compute nodes
  - Manages the debugger connections to the compute nodes
- Ratio of compute nodes to I/O nodes differs between machines
- All communication between CIOD and compute nodes is over virtual channel 0 of the tree network

CNK Function Shipping Example

- Application calls write() system call
- CNK breaks request into multiple messages
  - Size is configurable
  - Sends each message in turn to CIOD
- CIOD receives the message and calls write()
- CIOD sends result back to the compute node
- CNK collects the results from each message
- CNK returns result to application after either all of the data is sent or an error occurs
- CIOD never blocks on a system call
  - All sockets are implicitly non-blocking

CNK Function Shipping



Boot Process

- Control system starts microloader on compute nodes and I/O nodes
- Microloader boots CNK on compute nodes and Linux on I/O nodes
- Linux mounts file servers and starts CIOD
  - Customer can provide their own rc scripts
- CNK starts, initializes the compute nodes, sends message to CIOD
- CIOD starts, waits for all compute nodes to report, then waits for control system to connect

Job Startup

- CIOD accepts connection from control system
- Control system sends user login info and application info
- CIOD swaps to user
- CIOD reads application and sends to each compute node
- CNK loads application into memory and waits to start
- Control system sends start info
  - Debugger is optionally connected at start
- CIOD tells each compute node to start the application

Job Running and Termination

- CIOD forwards stdout and stderr text to control system
  - No support for reading from stdin
- Each compute node reports result to CIOD
  - Ended normally with exit status
  - Ended by signal with signal number
- CIOD forwards result to control system
- Control system waits for all compute nodes to end
- Control system closes connection to CIOD
- CIOD resets and waits for next job
- Control system can send signal to compute nodes
  - CIOD forwards to compute nodes

When things go wrong on I/O node

- CIOD is instrumented with trace points and status reporting
- If configured, CIOD listens on a service connection and supports commands to:
  - Turn tracing on and off
  - Report current status of compute nodes both summary and detailed
  - Report info about the tree network
- CIOD logs RAS events for error conditions
- CIOD tries to stay up and running even if an error occurs

CIOD Service Connection Example

```
telnet 172.30.60.152 7201
Trying 172.30.60.152...
Connected to 172.30.60.152.
Escape character is '^]'.
ciod running in coprocessor mode with 64 processors

> show_status
Mode: coprocessor
Job number: 1
Torus dimensions: X=8, Y=8, Z=8, T=1
Number of nodes: 64
Node   0:  state=RUNNING       , ioState=NOT_WAITING        , debug wait=NOT WAITING
Node   1:  state=RUNNING       , ioState=NOT_WAITING        , debug wait=NOT WAITING
Node   2:  state=RUNNING       , ioState=NOT_WAITING        , debug wait=NOT WAITING
Node   3:  state=RUNNING       , ioState=NOT_WAITING        , debug wait=NOT WAITING
Node   4:  state=RUNNING       , ioState=NOT_WAITING        , debug wait=NOT WAITING
Node   5:  state=RUNNING       , ioState=NOT_WAITING        , debug wait=NOT WAITING
Node   6:  state=RUNNING       , ioState=NOT_WAITING        , debug wait=NOT WAITING
Node   7:  state=RUNNING       , ioState=NOT_WAITING        , debug wait=NOT WAITING
```

When things go wrong on compute nodes

- CNK logs RAS events for error conditions
- If application dies, CNK creates a text “core” file with:
  - Register contents
  - Call stack
  - Interrupt history
- CNK monitors tree and torus networks and reports status at job end

CNK Summary

- CNK is a simple Linux-like kernel
  - Subset of system calls
  - Two modes of operation
- CIOD manages compute nodes and performs file I/O
- Job startup and termination is driven by the control system

Blue Gene/L Programming and Run-Time Environment

Peter Bergner

IBM Rochester

Outline

- Programming Environment
  - Differences from Linux/PPC
    - Unsupported syscalls
    - Syscalls with limitations
- Run-Time Libraries

Programming Environment

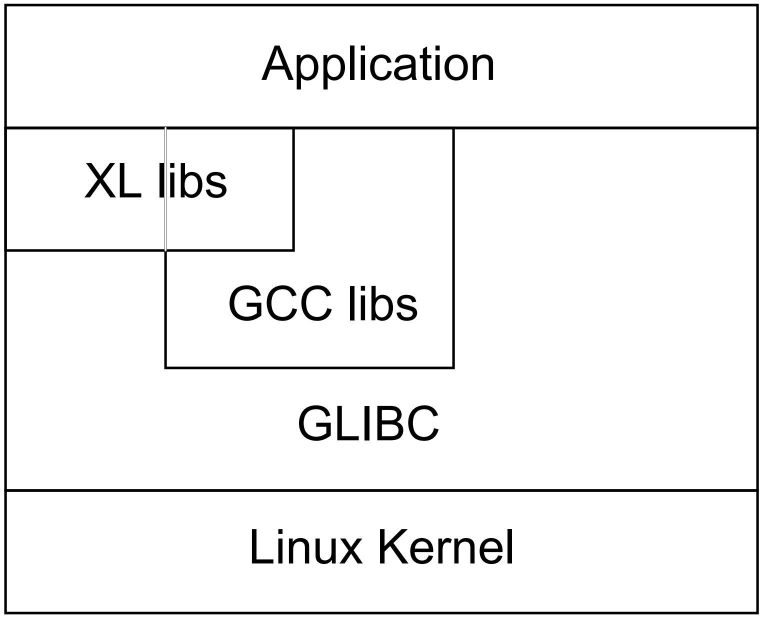
- Cross Compilation Environment
  - Front End Node running SUSE SLES9 Linux/PPC64
  - powerpc-linux-gnu -> powerpc-blrts-gnu
  - GNU toolchain for Blue Gene/L
  - IBM XL cross compilers for Blue Gene/L

Programming Environment cont.

- Similar programming model to Linux/PPC
- Differences from Linux/PPC:
  - No stdin
  - No asynchronous I/O
  - No dynamic linking
  - No demand paging / swap
    - virtual address space is mapped 1-1 with physical memory
  - No read only memory
    - due to CNK design decision
    - no SIGSEGV writing to a “const char *p”

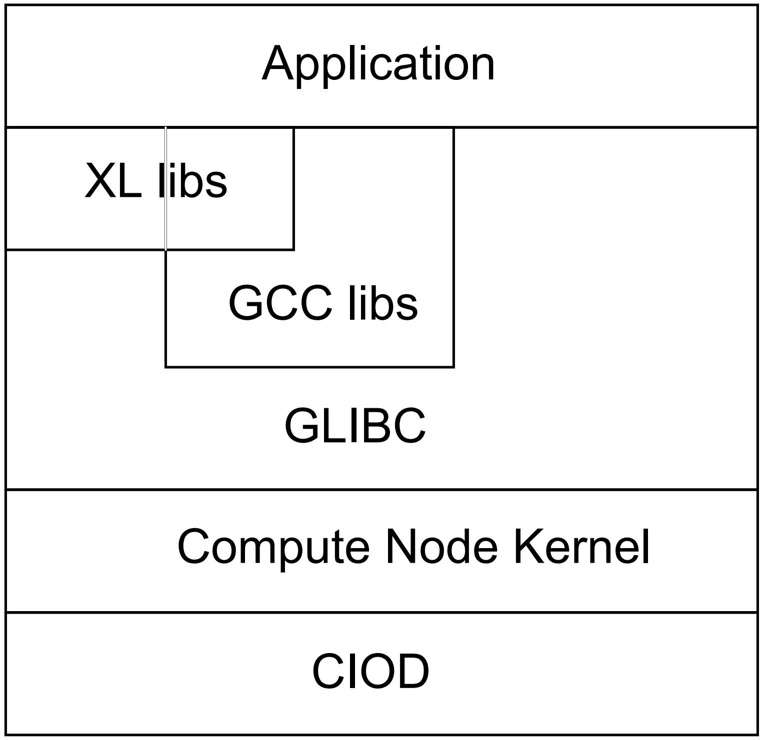
Programming Environment cont.

- Linux Software Stack



Programming Environment cont.

- Blue Gene/L Software Stack



Programming Environment cont.

- GNU Blue Gene/L toolchain
  - gcc, g++ and g77 v3.2
  - binutils (as, ld, etc.) v2.13
  - GLIBC v2.2.5
  - Blue Gene/L support supplied via patches
    - Customer applies the patches and builds the toolchain
    - IBM supplies scripts to download, patch and build everything

Programming Environment cont.

- IBM XL Compilers
  - Install IBM XLC V7.0 / XLF V9.1 compilers for SUSE SLES9 Linux/PPC64
  - Install Blue Gene/L add-on which adds Blue Gene/L versions of the XL run-time libs, compile scripts and config files.
    - Requires the GNU Blue Gene/L toolchain.
  - End result: Working Linux and Blue Gene/L compilers
    - Linux: xlc, xlC, xlf, xlf90, etc.
    - BG/L: blrts_xlc, blrts_xlC, blrts_xlf, blrts_xlf90, etc.

Programming Environment cont.

- IBM XL Compiler Options
  - -qarch=440 –qtune=440
    - Normal PowerPC FP code
  - -qarch=440d –qtune=440
    - Double Hummer FP code
  - -qhot=simd
    - Double Hummer FP code generated by TPO
  - -qipa
    - Interprocedural Analysis
  - -O4/-O5
    - Implicitly enable –qhot=simd –qipa
  - -qarch=auto, -qtune=auto, -qcache=auto
    - Disabled on Blue Gene/L

Unsupported Syscalls

- `fork, exec, clone, getppid, wait, waitpid`
- `mmap, mlock, madvise, mremap, msysnc, mprotect`
- `sigaction, sigprocmask, sigpending, sigsuspend, sigaltstack` (no POSIX signal handling)
  - We do support ANCI C signals.
- `capget, capset, getpriority, ioctl, ioperm, ipc, nice, prctl, ptrace`
- `chroot, mount`

Supported Syscalls With Limitations

- `kill(getpid(), signum)`
  - You can only send signals to yourself.
- `setitimer()`
  - You are allowed only one active timer.

Programming Environment cont.

- How to differentiate between AIX, Linux, Blue Gene/L in your code?

```
#if defined(__aix__)
    <aix code here>
#elif defined(__linux__)
    <linux code here>
#endif
```

Programming Environment cont.

- How to differentiate between AIX, Linux, Blue Gene/L in your code?

```
#if defined(__aix__)
    <aix code here>
#elif defined(__linux__)
    <linux code here>
#elif defined(__blrts__)
    <Blue Gene/L code here>
#endif
```

Programming Environment cont.

- How to differentiate between AIX, Linux, Blue Gene/L in your code?

```
#if defined(__aix__)
    <aix code here>
#elif defined(__linux__) || defined(__blrts__)
    <common linux and Blue Gene/L code here>
#endif
```

Run-Time Libraries

- GNU Run-Time Libraries
  - GCC libraries
    - GNU Standard C++ library (libstdc++.a)
    - GCC low-level run-time library (libgcc.a)
    - G77 run-time library (libg2c.a)
  - GLIBC libraries
    - GNU C library (libc.a)
    - Math library (libm.a)
    - IEEE floating point library (libieee.a)
    - G++ run-time library (libg.a)
    - Cryptography library (libcrypt.a)
    - NSS/Resolve libraries (libnss_dns.a, libnss_files.a, libresolv.a)

Run-Time Libraries

- IBM XL Run-Time Libraries
  - IBM C++ library (libibmc++.a)
    - Very light wrapper to libstdc++.a
  - IBM XLF run-time library (libxlf90.a)
  - IBM XL low-level run-time library (libxl.a)
  - IBM XL optimized intrinsic library (libxlopt.a)
    - Vector intrinsic functions
    - BLASS routines
  - IBM XL MASSV library (libmassv.a)
    - Vector intrinsic functions
  - IBM XL Open MP compatibility library (libxlomp_ser.a)

Questions?

Answers?

IBM Research

BlueGene/L IO Node Software Organization

C. Howson

Feb. 23, 2005

Outline

- Hardware Configuration
- Software Components on IO node
  - Kernel
  - Ramdisk
  - NFS mounted FS
  - BlueGene/L specific
- Performance
  - Network
  - File IO

IO Node Hardware

- IO nodes are same ASIC as Compute nodes, with different network connections wired
- ASIC has 4 network connections:
  - control, tree, torus, GigE
- Compute node has 3:
  - control, tree, torus
- IO node has 3
  - control, tree, GigE

- GigE network is only network suitable for high bandwidth IO

IO Node Software

- Linux kernel (2.4.19 + patches) on 1 CPU
- Minimal ramdisk based distribution (busybox)
- NFS mounted filesystem provides additional components (/bgl)
- BlueGene/L specific ciod
  - job management
  - system call functions for CNs

Boot Process

- Inital ramdisk and kernel image loaded via control network
- Kernel start message via control network
- Kernel boots
- Init starts
- RC scripts
  - /bgl NFS mount
- Extra kernel modules loaded
- Ciod starts
- Ready to accept jobs

IO node output of 'ps ax' after boot

PID	TTY	Uid	Size	State	Command
1	0		980	S	init
2	0		0	S	[keventd]
3	0		0	S	[ksoftirqd_CPU0]
4	0		0	S	[kswapd]
5	0		0	S	[bdflush]
6	0		0	S	[kupdated]
26	0		0	S	[rpciod]
68	1		1468	S	/sbin/portmap
72	0		6620	S	/sbin/ciod.440
80	0		996	S	/bin/sh
81	0		988	R	ps ax

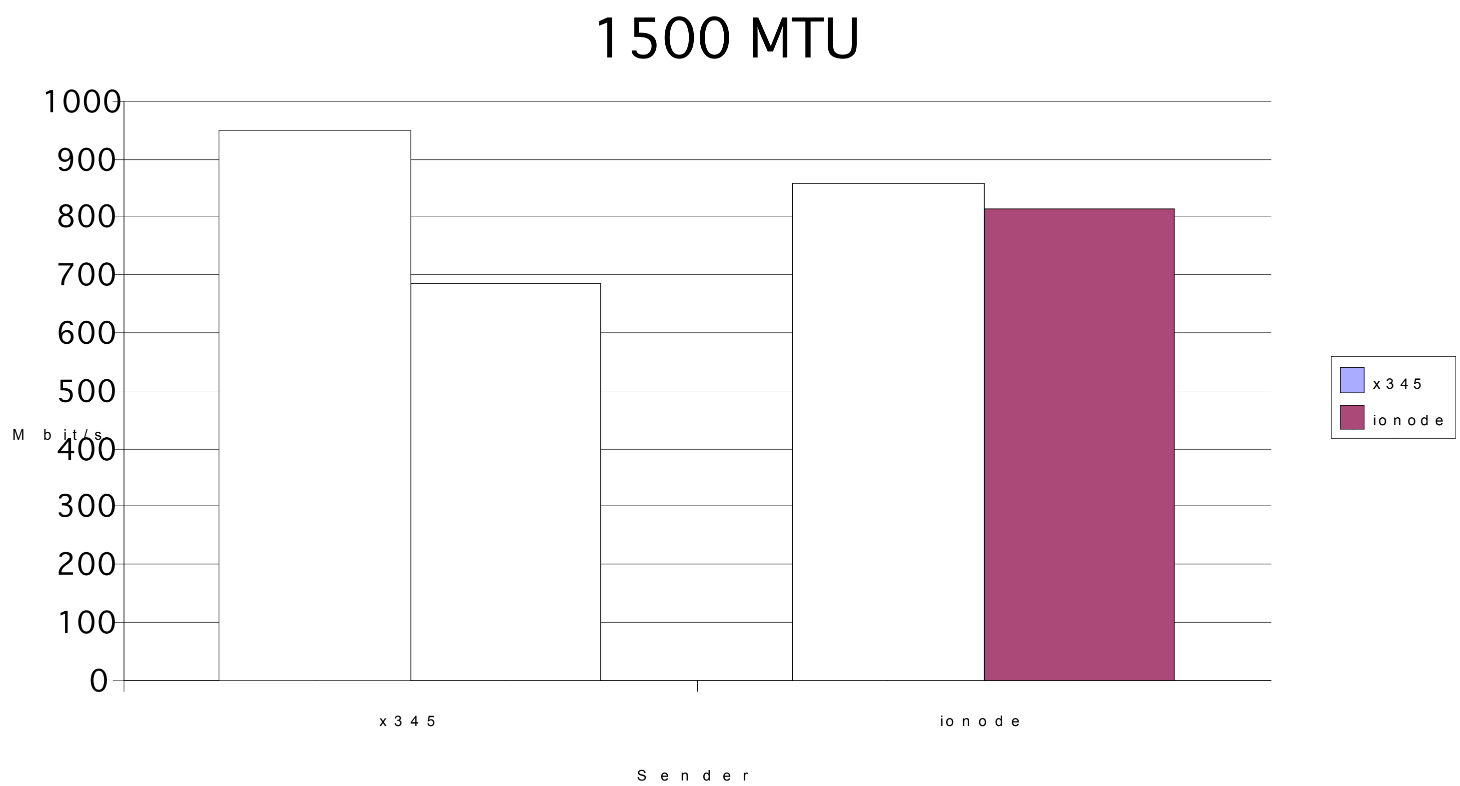
Design Philosophy

- IO node is more like an embedded system than general purpose computer
- Prefer static to dynamic memory allocation
  - CIOD is a single process: don't even fork for a new job
  - CIOD has relatively small fixed size buffers/compute node
- Avoid many daemons
  - consume memory
  - slow down IO due to scheduling conflicts
  - CPU is not fast wrt GigE
- Simple design
  - not fancy, but works
- Site specific configuration possible by modifying rc scripts
  - But try to offload functionality to external servers

Network performance: key to file IO performance

- Factors: MTU, sysctl, switch
- MTU: GigE supports Jumbo frames - up to 9000 vs standard 1500 bytes
  - Every host on physical net must be configured to accept jumbo frames
  - Private network helps for this
- sysctl: Linux default tcp settings are conservative (slow for GigE)
  - Modify apps to customize settings or change defaults
  - Set good defaults: /proc/sys/net/core/{r,w}mem_default
  - I like receive buffers to be 2x send buffers (512k,256k)
  - ifconfig txqueuelen 10000 helps a bit
- Network Switches are not crossbars
  - Keep IO nodes close to File servers on switch
  - Physical wiring choice affects performance

Netperf performance

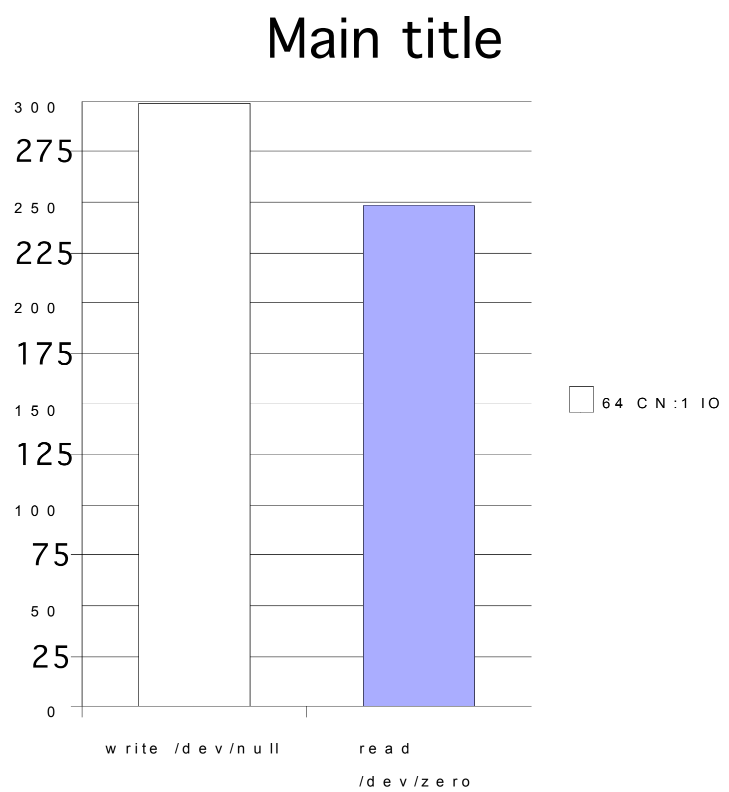


File IO Performance

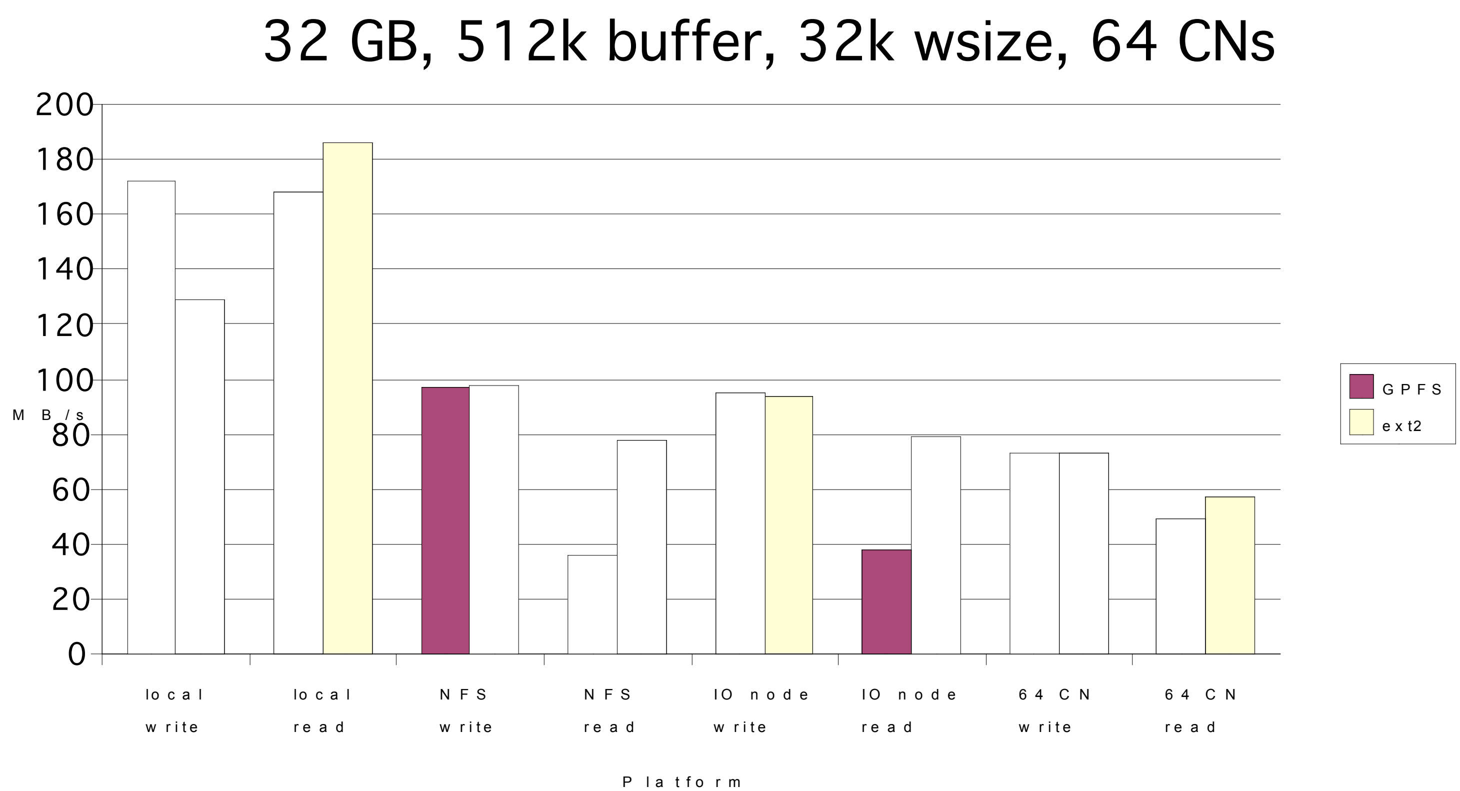
- IO node has very little RAM compared to aggregate compute node ram
  - Not so much opportunity to cache files on IO node
- Primary concern is streaming large files out to file servers
- Bottleneck is ciod/linux-fs/nwk, not CNK/tree/ciod
  - IO node software environment must be light and fast

CNK/tree/CIOD Performance

- Tree max theoretical bandwidth is 350 MB/s
- Measure by writing to /dev/null, reading from /dev/zero
- Read involves kernel zeroing buffers: 50 MB/s slower



GPFS and Ext2 Performance



Conclusion

- IO node software is very simple
- Filesystem and networking overhead is quite high
  - Be careful when adding daemons

PVFS2 and Parallel I/O on BG/L

Rob Ross
Mathematics and Computer Science Division
Argonne National Laboratory

Special acknowledgements

- Rob Latham – did most of the work to get PVFS2 up and running
- Susan Coghlan – provided all the access we needed, made everything easy for us, and clicked the mouse at the right time (even from SLC!)
- Kazutomo Yoshii – figured out how to get things built for the IO nodes at Argonne
- LLNL group (Robin, Ira, others) – provided us with a great start for building IO node kernels
- IBM – provided source to key components and insight into system components that made this possible

Outline

- PVFS2 introduction and background
  - What it is, who it is, and why it's interesting for BG/L
- Base functionality for PVFS2 on BG/L
  - What is working, preliminary performance numbers
- Beyond the baseline
  - Pursuing higher I/O performance
  - Research in MPI-IO
- Wrap up

The PVFS2 Parallel File System

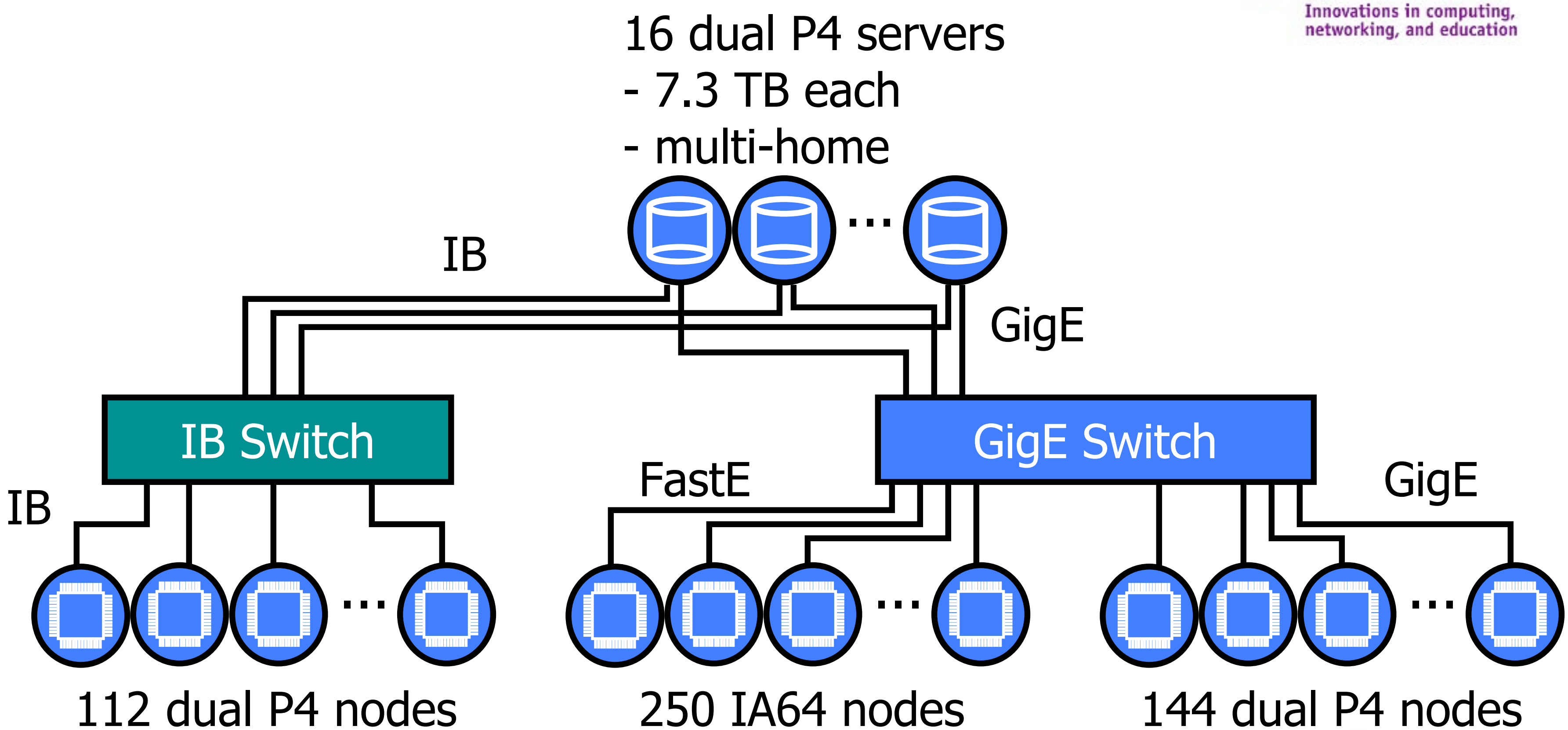
- Parallel file system
  - Distributed data and metadata
  - Tuned for performance and concurrency
- Production ready
  - In use at ANL, OSC, Univ. of Utah CHPC, others
- Open source and open development
  - LGPL license on all but kernel module, GPL on kernel module
  - Current CVS is anonymously accessible
  - Mailing lists where developers can track and initiate discussions
- Community research vehicle
  - Heterogeneous system support
  - Predominantly user-space code
  - Rapid porting via network and storage abstractions
  - Many labs and universities extend or modify PVFS2 to explore new ideas

Who is PVFS2?

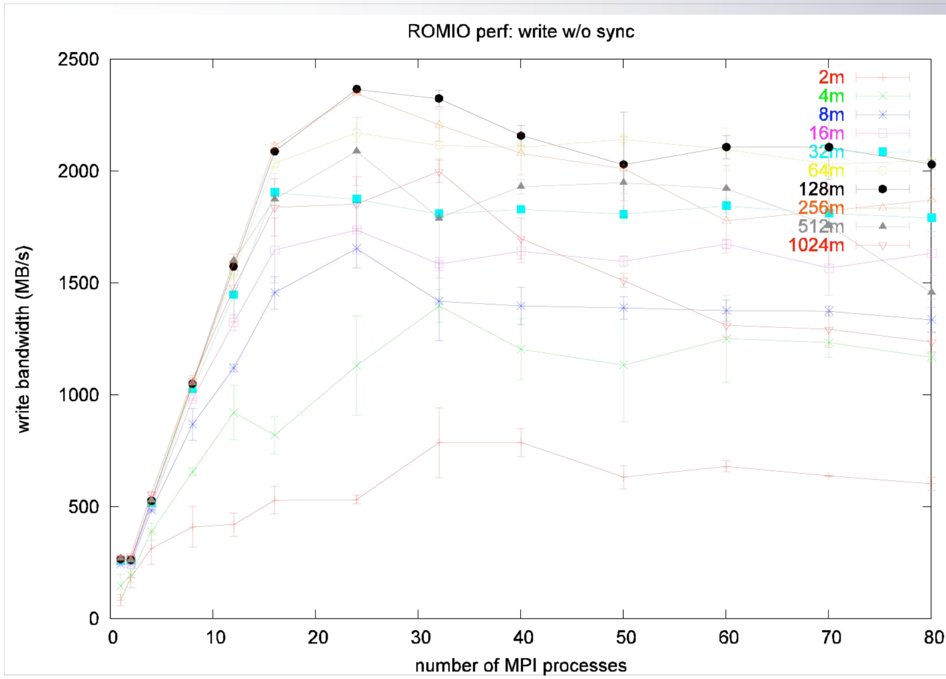
- PVFS2 is an open, collaborative effort
- Core development
  - Argonne National Laboratory
    - Ross, Latham, Gropp, Thakur
    - Supported by DOE Office of Science
  - Clemson University
    - Ligon, Settlemyer
  - Ohio Supercomputer Center
    - Wyckoff, Baer
- Collaborators
  - Northwestern University
    - Choudhary, Ching
  - Ohio State University
    - Panda, Yu
  - Penn State University
    - Sivasubramaniam, Kandemir, Vilayannur

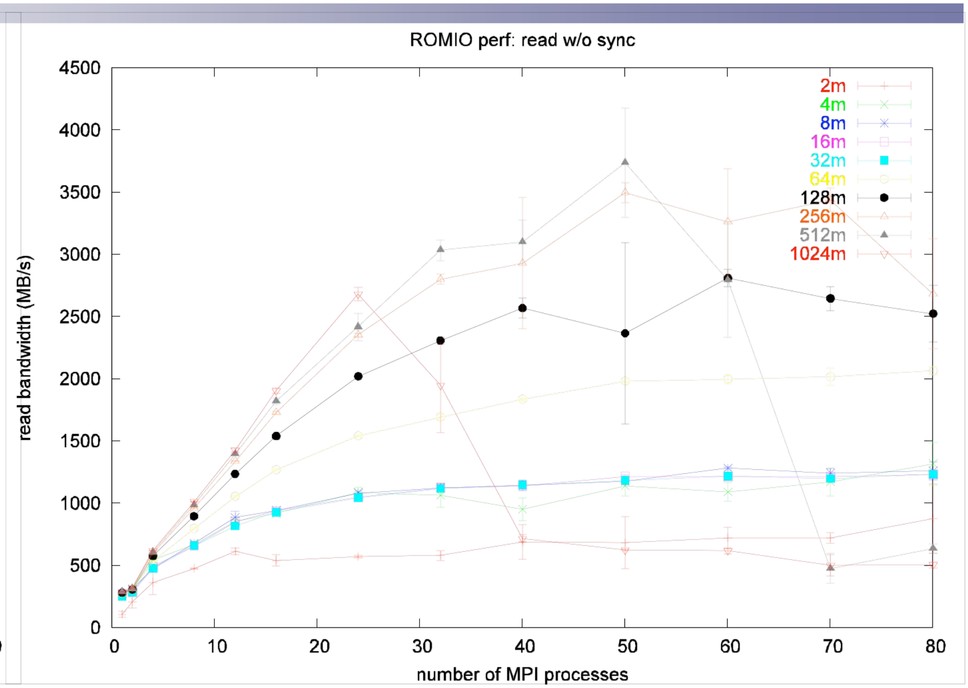
PVFS2 at OSC

- 506 total clients
- 116.8 TByte file system



OSC cluster performance

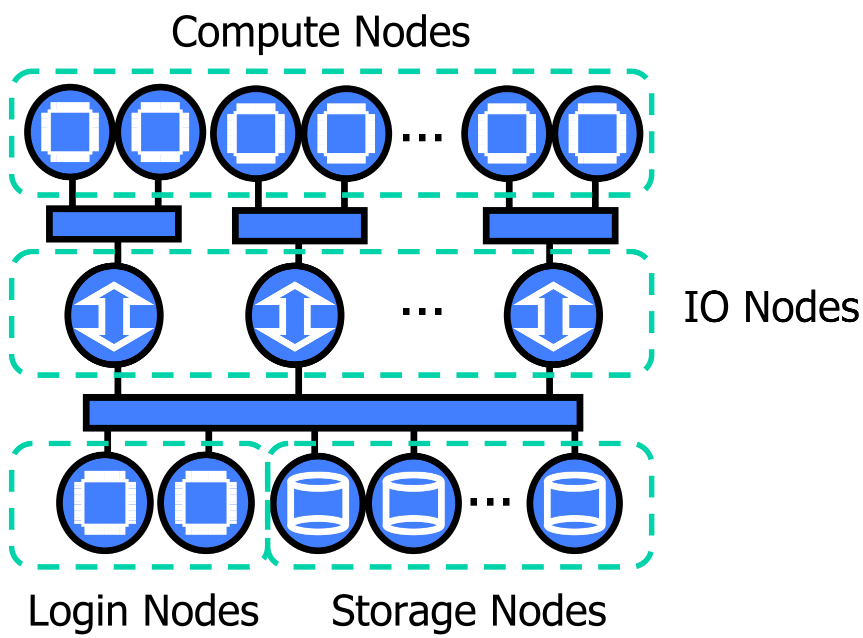




- Data sizes are per-client
- Achieving ~2.8GB/sec write, ~3.8GB/sec read
  - No network optimization (memory registration or pipelining)

View of I/O on BG/L

- Storage nodes
  - Local access to disks
  - GigE connections to login and IO nodes
- Login nodes
  - Interactive machines
  - <u>Place where data staging will occur</u>
- IO nodes
  - Aggregators for compute node I/O
    - 1:8 to 1:64 ratio of IO nodes to compute nodes
  - Tree connection to compute nodes
- Compute nodes
  - <u>Source/sink of runtime I/O</u>

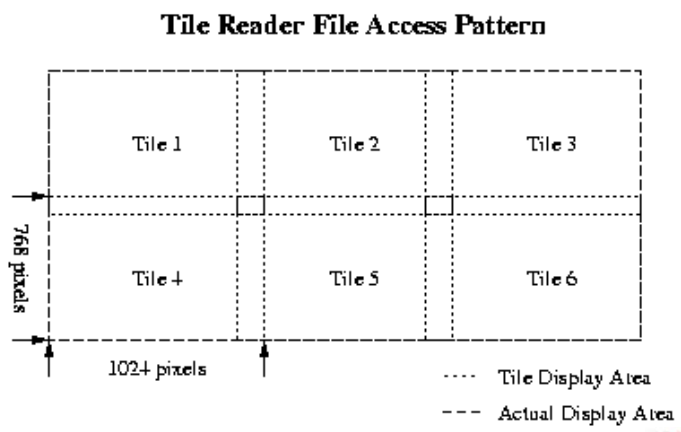


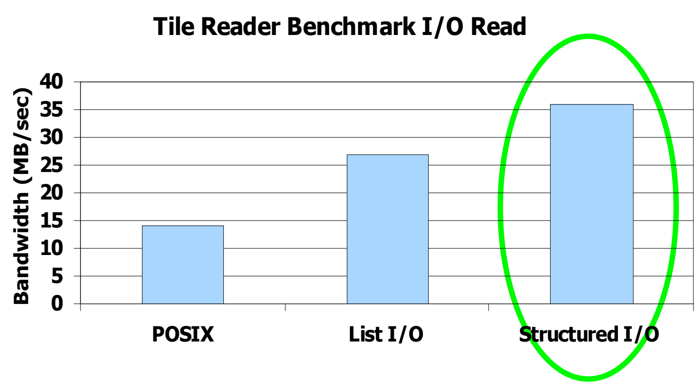
Why put PVFS2 on BG/L?

- It's fun ☺
- It provides another data point for I/O performance
- Most importantly, PVFS2 addresses three key scalability problems for parallel file systems:
  - I/O performance (especially for noncontiguous data)
  - Metadata performance (in particular open/close)
  - Failure tolerance
- Because of these advantages, we believe that PVFS2 has the best chance of extracting the highest possible I/O performance from BG/L

Scaling effective I/O rates

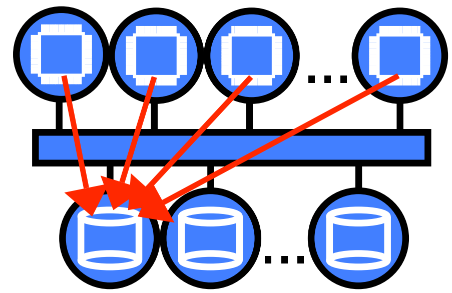
- POSIX I/O APIs aren't descriptive enough
  - Don't allow us to generally describe noncontiguous regions in both memory and file
- POSIX consistency semantics are too great a burden
  - Require too much additional communication and synchronization, not really required by many HPC applications
  - Will never reach peak I/O with POSIX at scale, only penalize the stubborn apps
  - Use more relaxed semantics at the FS layer as the default, build on top of that



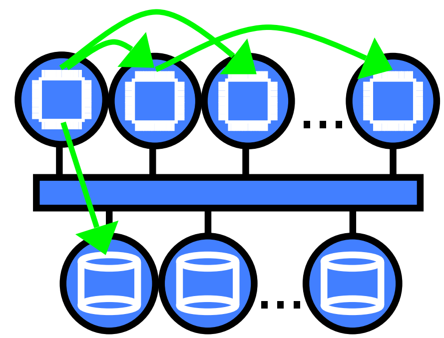


Scaling metadata operations

- POSIX API hinders scalability here too
  - POSIX open/close access model imposes constraints on how we implement MPI-IO operations like MPI_File_open
  - Similar issues with fsync and other operations

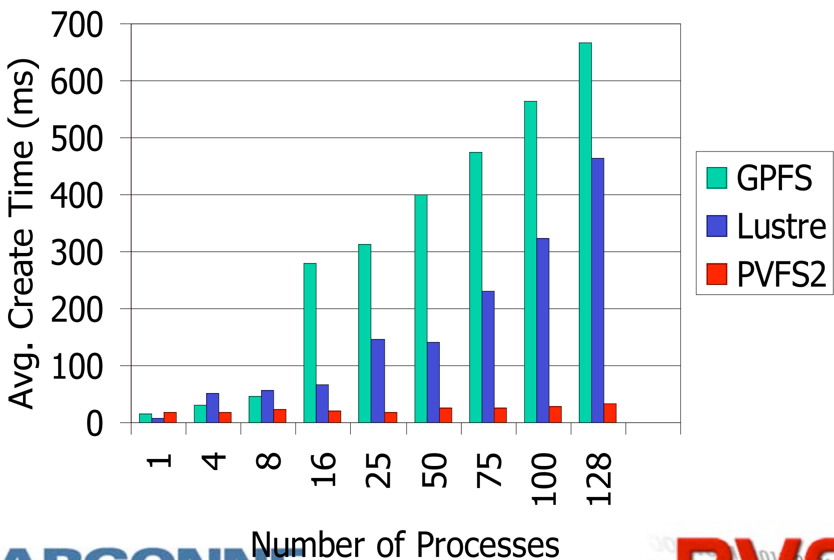


POSIX file model forces all processes to open a file, causing system call storm.



Handle-based model uses a single FS lookup followed by broadcast of handle (implemented in ROMIO/PVFS2).

MPI File Create Performance (small is good)



ARGONNE NATIONAL LABORATORY

PVFS2 PARALLEL VIRTUAL FILE SYSTEM

Office of Science U.S. DEPARTMENT OF ENERGY

Tolerating client failures

- Client failures are likely to be common with high node counts
  - 99.99% up indicates ~6 nodes down at any time on a 64K node system
  - 99.9% up indicates ~65 down at any time on same
- Unlike other options, PVFS2 uses a **stateless I/O model**
  - No locking system to add complications
  - No other shared data stored necessary for correct operation (no tracking of open files, etc.)
- Client failures can be ignored completely by servers and other clients
  - As opposed to locking systems, where locks and dirty blocks must be recovered!
- Server restarts are easily handled as well

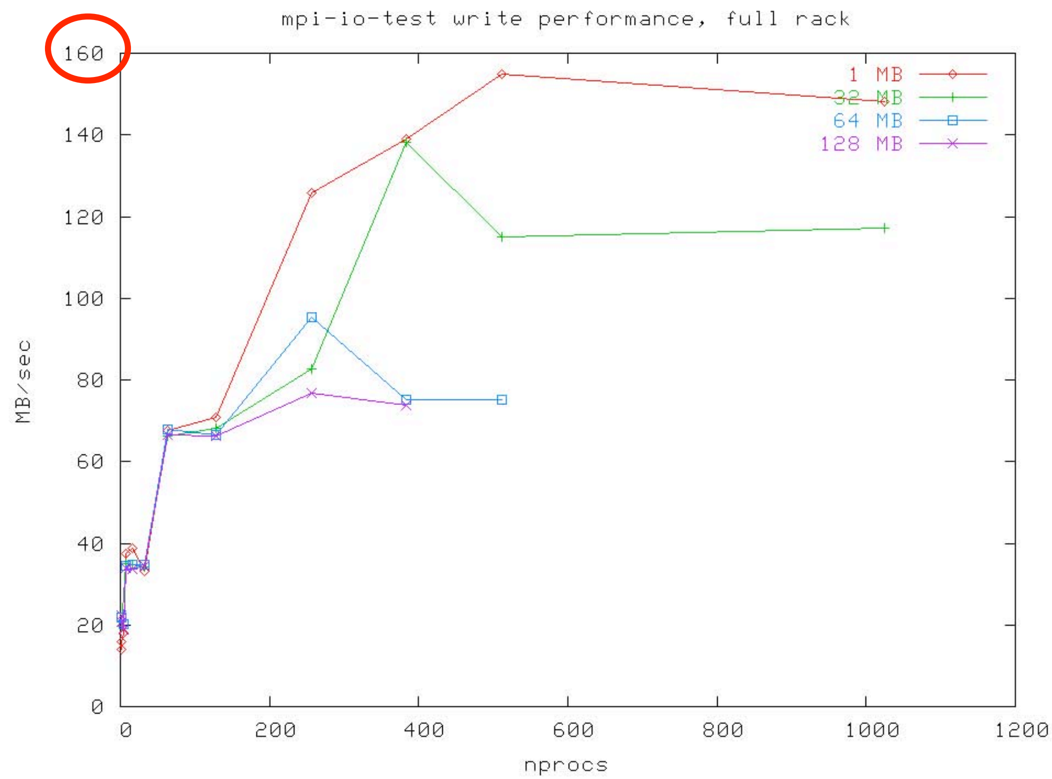
First steps in running PVFS2 on BG/L

- **Goal: Enable data staging and runtime I/O to a PVFS2 file system**
  - Run PVFS2 servers on storage nodes
    - dual Xeon nodes running SLES Linux and 2.6.5 kernel
  - Mount PVFS2 file system on login nodes
    - PowerPC 970 nodes running SLES Linux and 2.6.5 kernel
  - Mount PVFS2 file system on IO nodes
    - BG/L PowerPC nodes running 2.4.19 kernel (no longer MontaVista)
- **This only took two weeks to accomplish!**
  - Mostly learning/creating build environment
  - Minimal patching to PVFS2 (all in CVS)
  - 12 PVFS2 servers providing a single coherent file system

    (Assuming 900mbit/sec network to each, peak of ~1.3GB/sec raw BW)

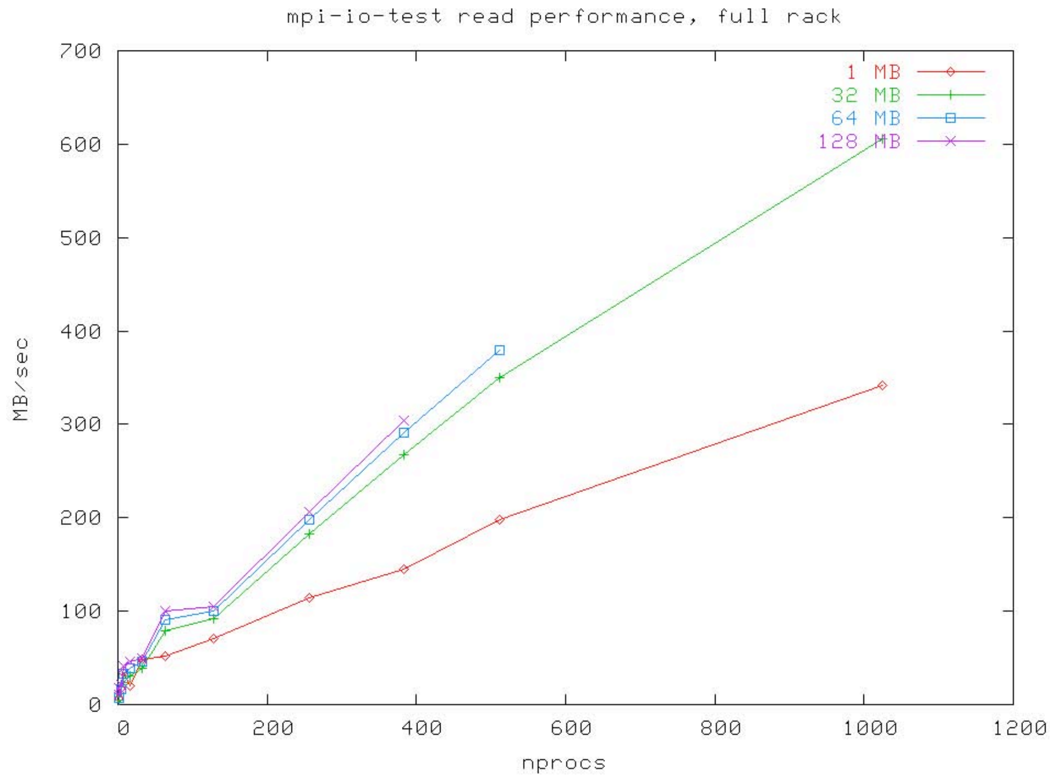
Write performance (the bad news)

- Simple pattern:
  - Single file
  - Independent MPI-IO
  - One big access each
- We have more work to do here!
- ciod is breaking accesses into 95520 byte blocks
  - Understand why better now (Mike's talk)
  - Try tuning I/O message size
    - **What's the variable?**
  - Check TCP buffer sizes and turn on jumbo frames (Chris's talk)
  - Why does strace'ing the ciod kill our machine sometimes?
  - "Happy SuperComputing!" to you too ☺



ARGONNE NATIONAL LABORATORY · PVFS2 PARALLEL VIRTUAL FILE SYSTEM · Office of Science U.S. DEPARTMENT OF ENERGY

Read performance (the good news)



- Peak of 600MB/sec (44% of raw BW, also no tuning)
  - This is **with** those tiny blocks…

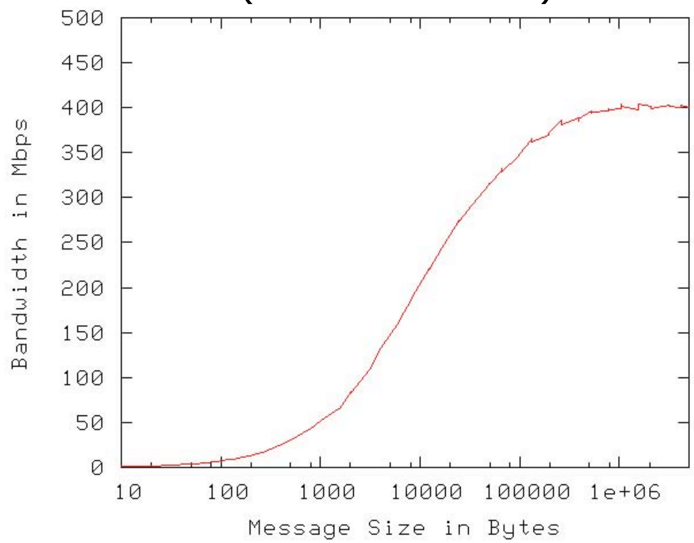
Beyond base functionality

- Our research indicates that the POSIX interface limits I/O scalability
  - Noncontiguous read and write performance
  - Open/close problems
- PVFS2 improvements cannot be seen through the VFS interface
  - BG/L has already broken POSIX, so on the right path…
  - We're still going through the VFS
  - The ciod is using POSIX calls
- **To obtain the highest possible performance we must circumvent (or change!) the VFS**
- Two options:
  - Direct compute node to storage server communication
  - **Retool communication between compute and IO nodes and mechanism IO node uses to access file system**

Direct PVFS2 access from compute nodes

- **Idea: Use PVFS2 client library directly on top of socket call forwarding to bypass IO node mount point**
- BGL PowerPC nodes
  - Special, proprietary kernel
  - Not all system calls are forwarded
- PVFS2 client code will (now) build for compute nodes, but
  - poll() and select() aren't implemented, so we can't run
- Interesting experiment, but not ideal solution…

TCP Bandwidth over Tree
(modified NetPIPE)



Changing the I/O language

- **Really what we'd like to do is change how compute processes talk to the file system**
  - Ideas prototyped in PVFS2 already
  - Allow for efficient noncontiguous I/O
  - Eliminate open() and close() scalability issues
  - More efficiently leverage the tree, IO node, and GigE
- This means changing how compute processes communicate with the IO node
  - Replace or augment existing ciod functionality
  - Map new language to PVFS2, GPFS, Lustre operations
    - **These changes can benefit any underlying file system**

I/O research in BG/L

- **Plan:** Experiment with new MPI-IO algorithms
  - Control of access mapping to IO nodes
  - Caching of data at IO nodes (to what extent possible)
  - New GPFS, Lustre, PVFS2 optimizations
- To do this, we must be able to rebuild ROMIO and link to IBM MPI
- **Next Tasks:**
  - ANL ensures that ROMIO builds cleanly against IBM MPI
  - IBM provides MPI without ROMIO

Wrap up

- In a couple of weeks we were able to get PVFS2 running on BG/L
  - Open source operating systems played a key role
  - Very positive experience!
- IBM developers have been very helpful
  - Will aid greatly in MPI-IO research and tuning for BG/L
- This is turning into an ideal platform for testing and deployment of next-generation I/O systems!
- High level libraries will follow as well
- We could use just a little more source… ☺

Additional information on PVFS2

- PVFS2 web site: http://www.pvfs.org/pvfs2
  - Documentation, mailing list archives, and downloads
- PVFS2 mailing lists (see web site)
  - Separate users and developers lists
  - Please use these for general questions and discussion!
- Email
  - Rob Ross <rross@mcs.anl.gov>
  - Rob Latham <robl@mcs.anl.gov>

ARGONNE NATIONAL LABORATORY

PVFS2 PARALLEL VIRTUAL FILE SYSTEM

Office of Science U.S. DEPARTMENT OF ENERGY

BG/L Control System Software

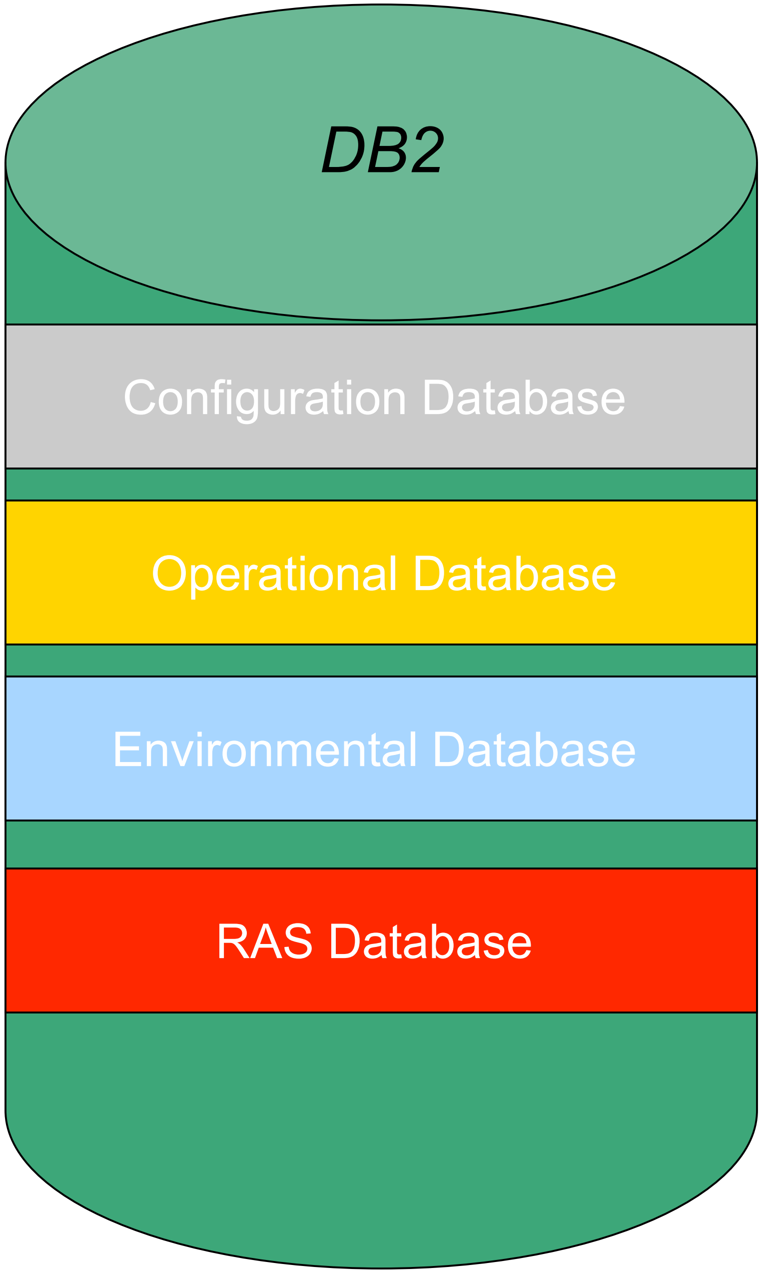
Mark Megerian
IBM Rochester

February 2005 | Blue Gene/L

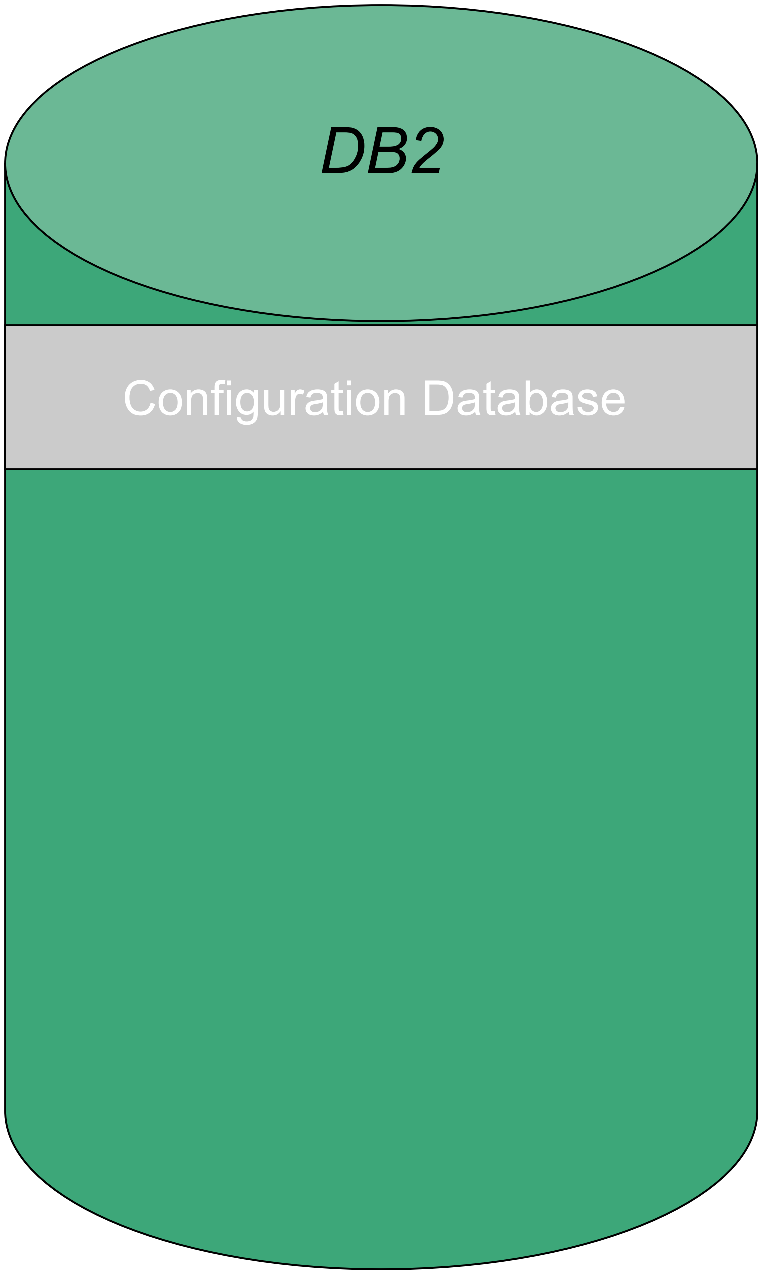
BG/L Control System Overview

- Control system software runs on the service node
- Control system software manages the aspects of system operation up through job submission onto the BG/L core
  - Hardware discovery
  - System partitioning
  - Booting of partitions
  - Job submission / job polling
  - RAS data collection
  - Hardware monitoring
- DB2 is the central repository of all control system information
  - Allows control system components to get hardware information and topology from the database, which is always kept current
  - Less direct contact with the hardware

BG/L DB2 Structure

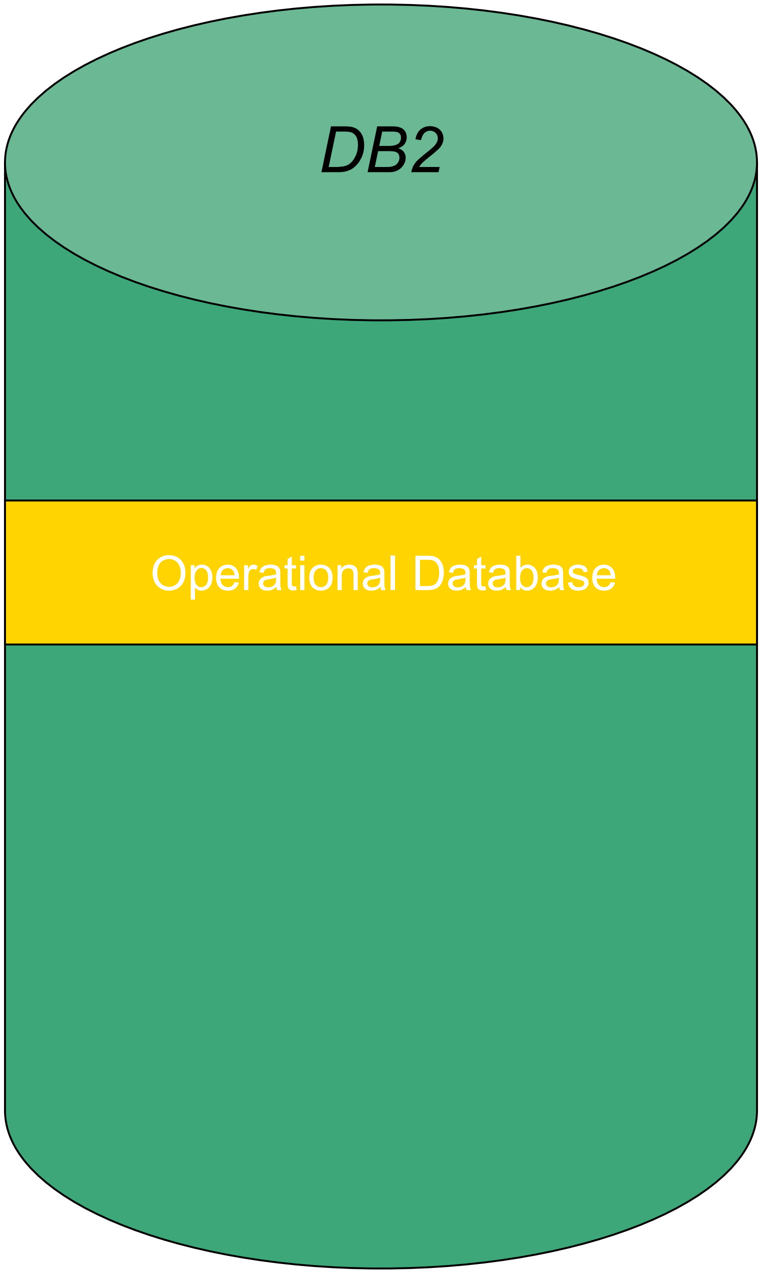


- Configuration database is the representation of all the hardware on the system
- Operational database contains information and status for things that do not correspond directly to a single piece of hardware such as jobs, partitions, and history
- Environmental database keeps current values for all of hardware components on the system, such as fan speeds, temperatures, voltages
- RAS database collects hard errors, soft errors, machine checks, and software problems detected from the compute complex.



BG/L DB2 Structure

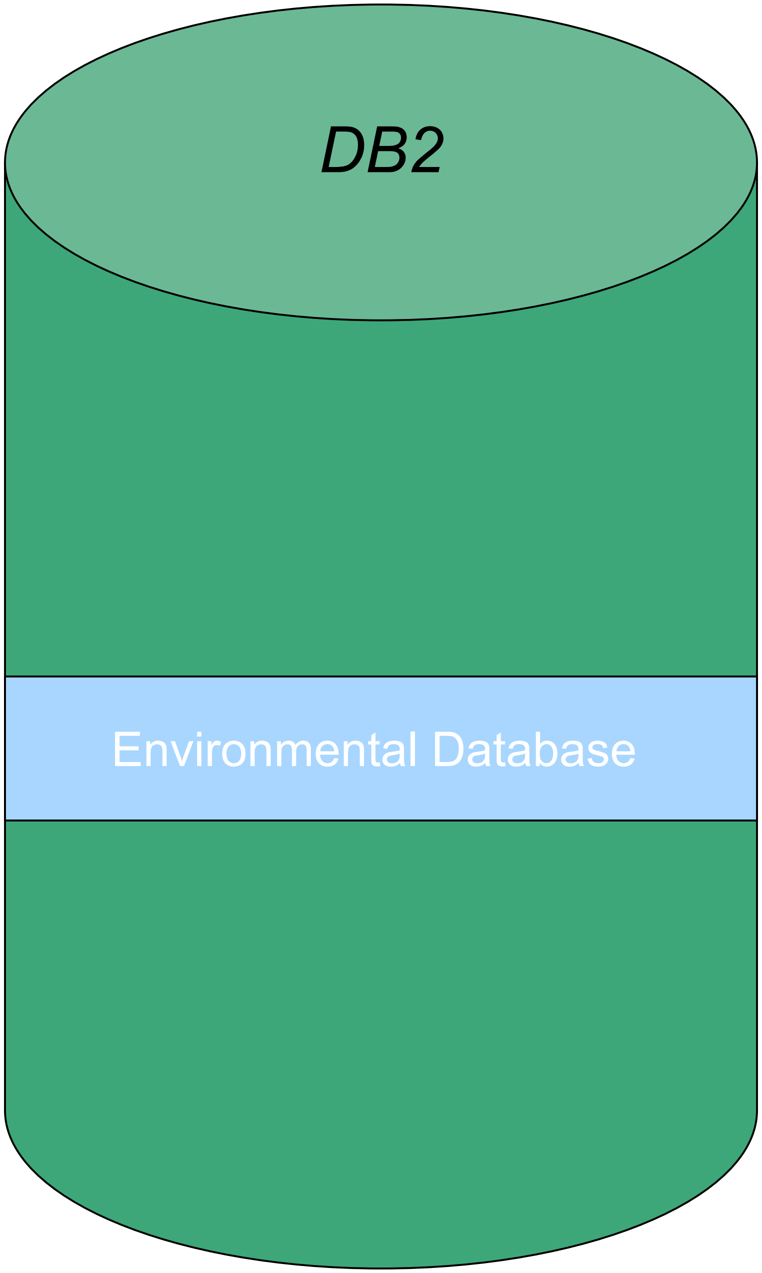
- Configuration database is the representation of all the hardware on the system
  - Machine
  - Midplanes
  - Service Cards
  - Link Cards
  - Link Chips
  - Node Cards
  - Processor Cards
    - Compute & I/O
  - Nodes
  - Cables
  - Ido Chips
  - Clock Cards
  - Fan Modules



BG/L DB2 Structure

- Operational database contains information and status for things that do not correspond directly to a single piece of hardware such as jobs, partitions, and history

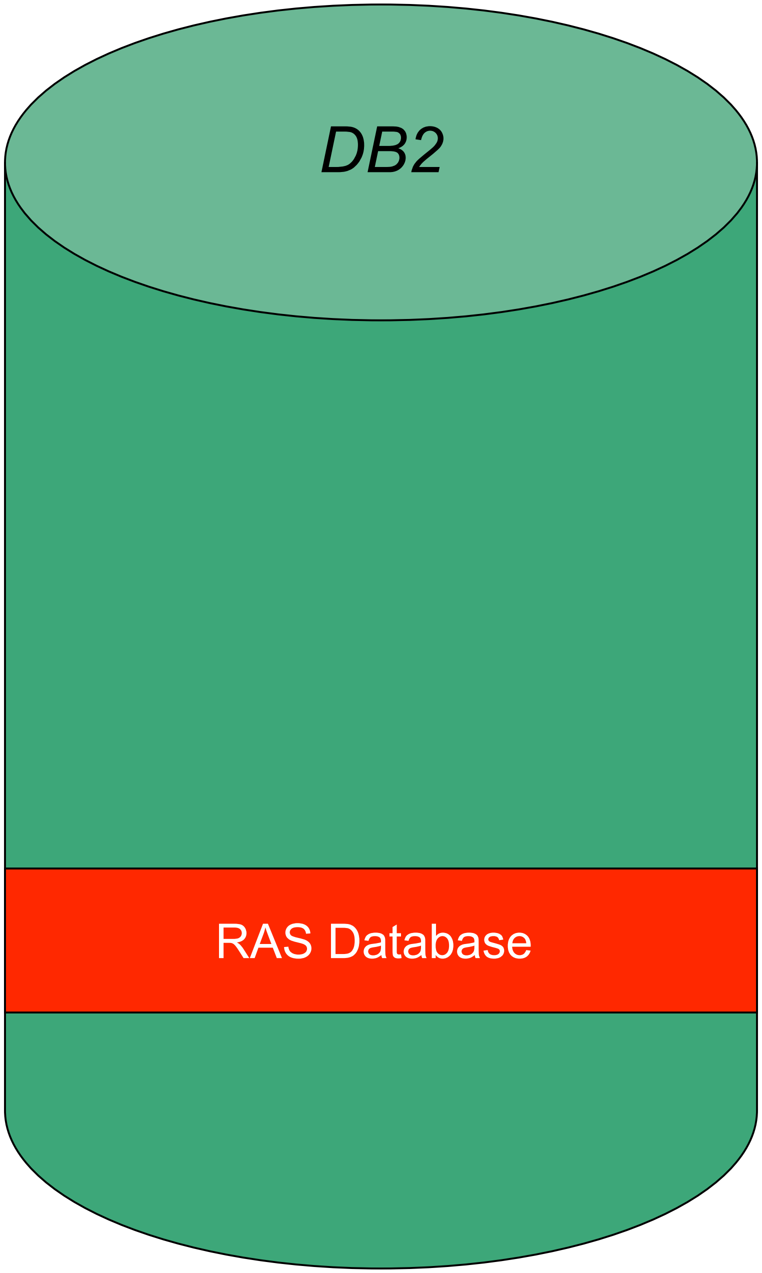
  - Blocks (partitions)
  - Jobs
  - Job history
  - Switch settings
  - Link <-> Block map
  - Block users



BG/L DB2 Structure

- Environmental database keeps current values for all of hardware components on the system, such as fan speeds, temperatures, voltages
  - Fan Modules
    - Desired RPMs
    - Actual RPMs
    - Voltages
    - Temperatures
  - Service Cards
    - Ambient temp
    - Chip temps
    - Voltages
  - Node Cards
    - Chip temps
    - Temp limits
    - Wiring faults
  - Link Cards
    - Power Status
    - Temps

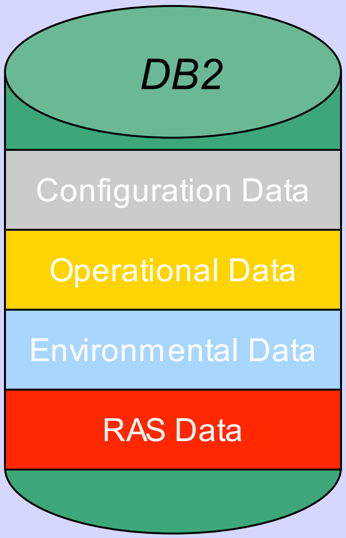
BG/L DB2 Structure



- RAS database collects hard errors, soft errors, machine checks, and software problems detected from the compute complex.
  - RAS events collected by Discovery for bad hardware, missing cards, bad memory, bad cables
  - RAS events collected from compute complex while jobs are running, from kernel interrupts
  - RAS events generated by HW monitoring, for wiring faults, bad cards, fan speeds, over temps
  - RAS events generated by MMCS during link training, software errors, file system errors

Control System Components

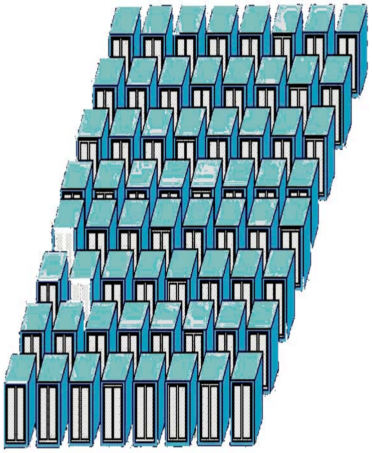
BG/L Service Node



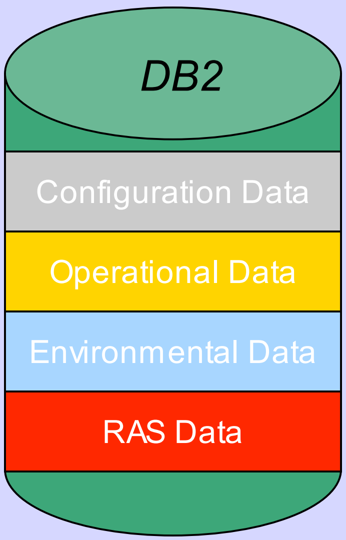
- Proxy
- Discovery
- MMCS
- RAS
- Monitor
- CIODB

All are controlled by a “master” process that watches each process, and restarts upon failure.

All interact directly with the database, and interact with BG/L core.



Control System Components



Proxy

Discovery

MMCS

RAS

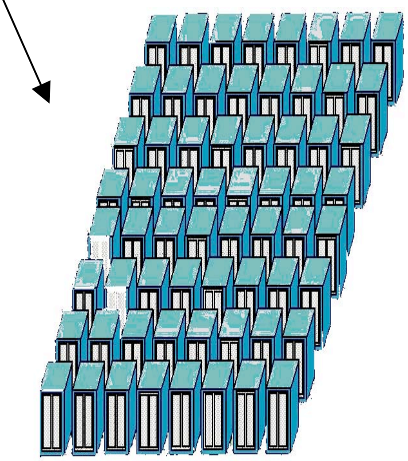
Monitor

CIODB

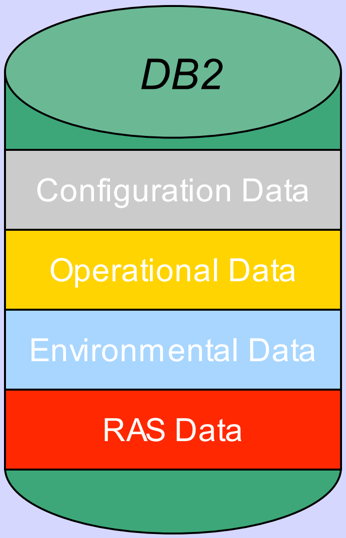
Always running on service node.

Marshals traffic to the hardware, and blocks multiple connections to the same hardware.

Accesses the database for configuration data



Control System Components



Proxy

Discovery

MMCS

RAS

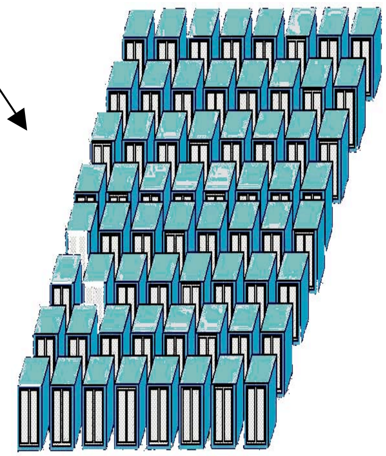
Monitor

CIODB

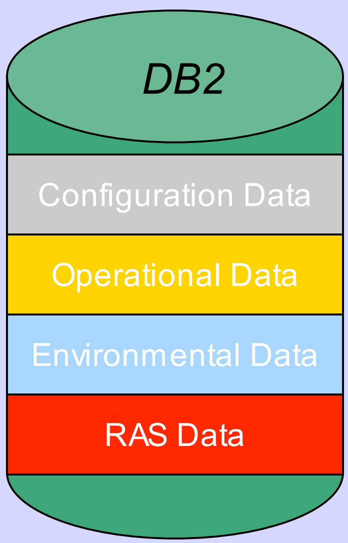
A process that is used during initial setup that discovers all the hardware on the system and populates the database with the configuration info.

Discovers cables between midplanes.

Runs beyond initial setup to flag hardware that goes offline, or new hardware that is added.



Control System Components



Proxy

Always running, and polling the database.

Discovery

MMCS Server polls for tasks submitted by mpirun, mmcs consoles, and by scripts

MMCS

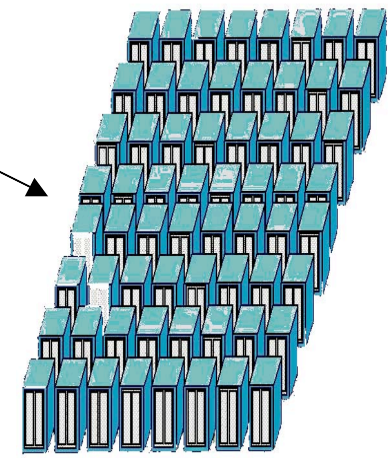
MMCS also spawns threads for monitoring each block, once its been booted

RAS

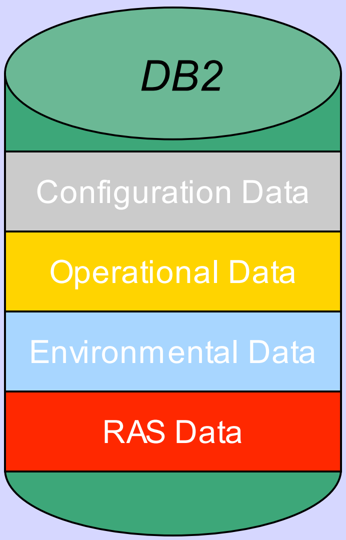
Monitor

There is an MMCS console that can be used to directly interact with MMCS

CIODB



Control System Components



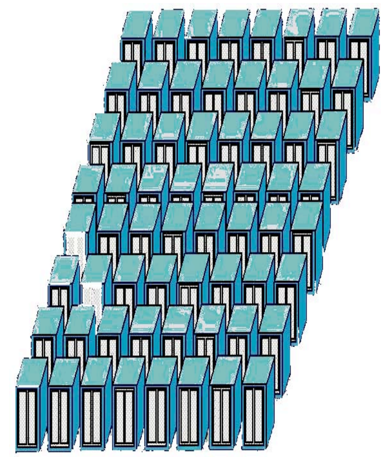
- Proxy
- Discovery
- MMCS
- **RAS**
- Monitor
- CIODB

Reliability / Availability / Serviceability Data

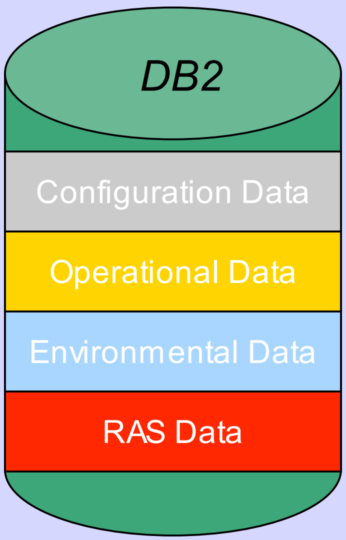
Collected from nodes during job execution

Also collected during other phases, such as booting blocks, discovering hardware, monitoring hardware

All RAS events stored with location of event, timestamp, block, job, node number, event text



Control System Components



Proxy

Discovery

MMCS

RAS

Monitor

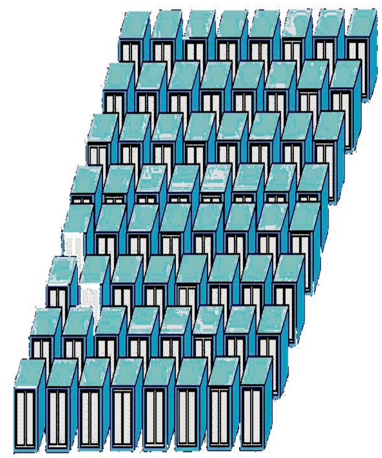
CIODB

Runs all the time, or on an "as-needed" basis.

Monitors individual cards for temperatures, fan speeds, voltages, power supply status, etc.

Console for controlling all the monitoring, and viewing the data that has been collected.

Runs against node cards, link cards, and service cards.



Control System Components

Proxy

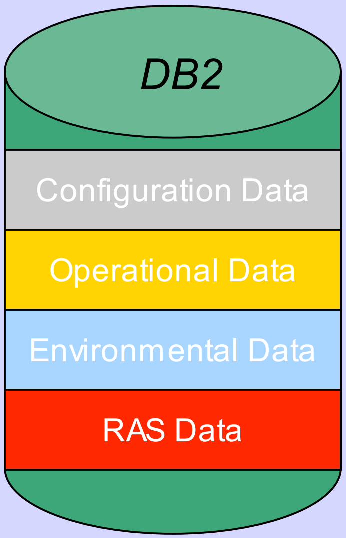
Discovery

MMCS

RAS

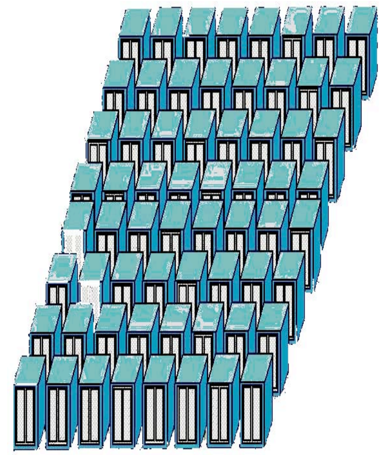
Monitor

CIODB



Ciodb polls the database for job requests, such as starting jobs and killing jobs

Ciodb spawns a thread for each submitted job to poll for output, and for completion



Life Cycle of a Partition

- Partition is defined as a collection of midplanes and switch settings, and implied cables
- Partition that is defined, starts in FREE status, and contains the following information:
  - Block ID (partition name)
  - Size (number of midplanes)
  - Shape (x, y, z dimensions)
  - Torus or Mesh
  - Mode (co-processor mode, virtual node mode…)
  - Ratio of IO nodes to compute nodes
  - Path for Microloader image
  - Path for RAMdisk image
  - Path for Linux kernel image
  - Path for CNK image
  - Creation timestamp
  - Owner
  - Status
- Partitions can be defined using the console, with APIs using XML, by calling MPIRUN, or by using an external scheduler

Life Cycle of a Partition

- Partition is initially FREE
  - The process of booting the partition starts with an “allocate”
- Partition goes to ALLOCATED
  - The components of the partition are allocated, and therefore cannot be used by another partition
  - The ido connections are established for node cards and link cards via the proxy
  - The switch settings are made to program the link chips for either torus or mesh
- Partition goes to CONFIGURING
  - Microloader is loaded onto all nodes
  - RAMdisk is loaded onto IO nodes
  - Linux kernel is loaded onto IO nodes
  - CNK is loaded onto compute nodes
- Partition goes to BOOTING
  - All nodes are started, torus and tree links established
  - IO nodes mount the file system and establish ethernet connections
  - Ciod starts, and sends [ciod initialized] message to CIODB
- Partition goes to INITIALIZED when all IO nodes have responded
  - Jobs can be submitted
  - MMCS polls for RAS events
  - Freeing the partition takes it back to FREE, and ido connections are released

Life Cycle of a Job

- Job is created on the system in QUEUED status
- Job contains the following information
  - Path to executable
  - Partition name
  - User name
  - Arguments
  - Environment variables
  - Stderr and stdout file or redirection
  - Working directory
  - Status

Life Cycle of a Job

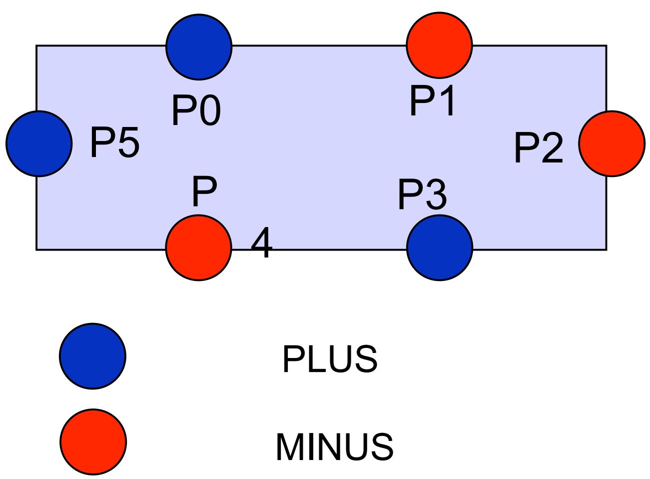
- Job is created on the system in QUEUED status
  - Arguments can be added at this point, if they weren't added at job creation time
- Startjob command or setJobState API call moves job to STARTING
  - This status value tells to ciodb to start the job, provided the block is not already busy running another job
- Ciodb contacts the IO nodes with the job and user information, starts the job and status moves to RUNNING
  - Ciodb polls the job for stderr, stdout, and completion
- When ciodb is told by ciod that the job has ended, job status goes to TERMINATED
  - Job record is moved from active job table to job history table
- If a user kills the job prior to completion, there is an intermediate status of DYING
  - This notifies ciodb to kill the job, and then set the status to TERMINATED

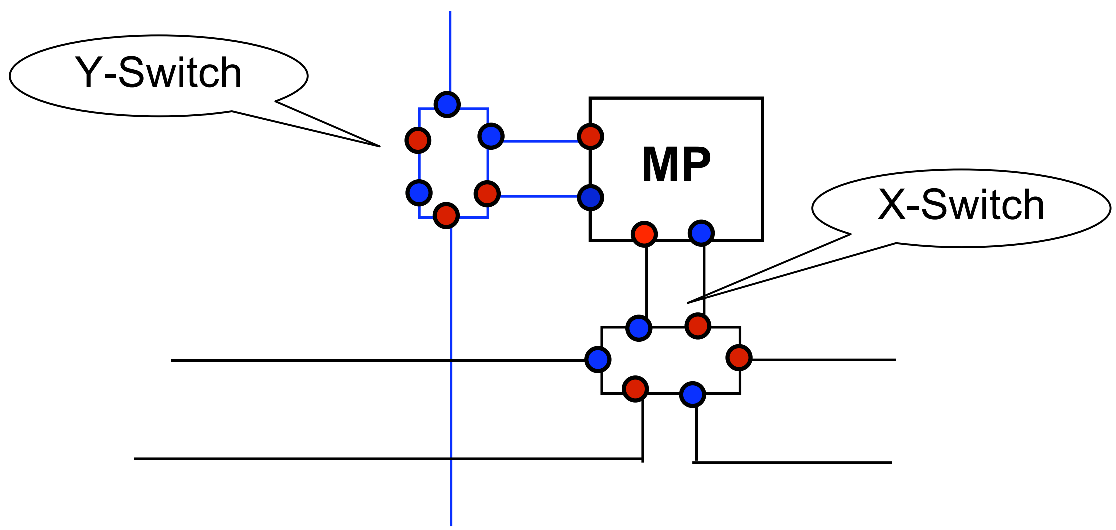
Partitioning of BG/L

- The control system manages all aspects of hardware partitioning, using cables and link chip programming
- The control system handles link programming by making “switch settings” during the boot process
- Allows many simultaneous jobs to be running
- Each partition is isolated
- Partitions can be running with different kernels

The Switches

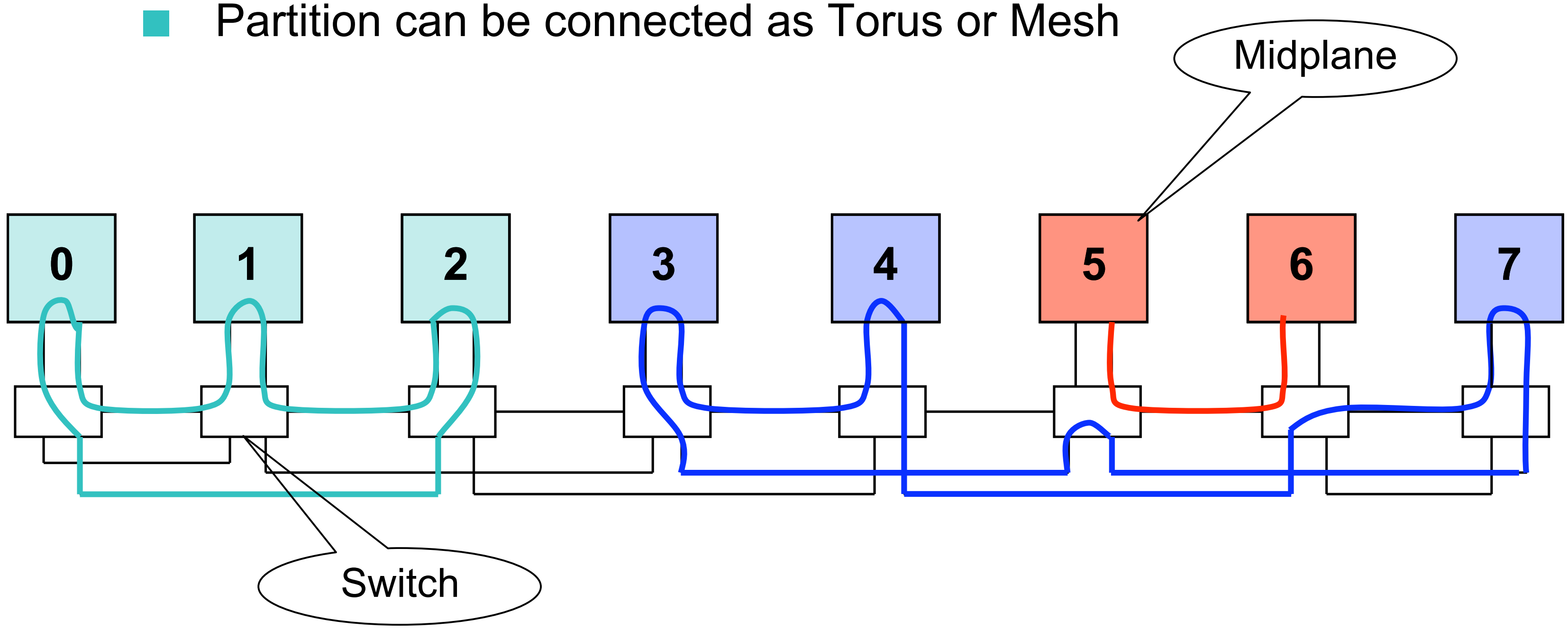
- Each Midplane is connected to three switches
  - One switch on each dimension (X/Y/Z)
- Each switch has six ports (P0..P5)
  - Two ports connect to the midplane (P0,P1)
  - Other four connect to other switches (P2..P5)
- No direct connection between switches on different dimensions



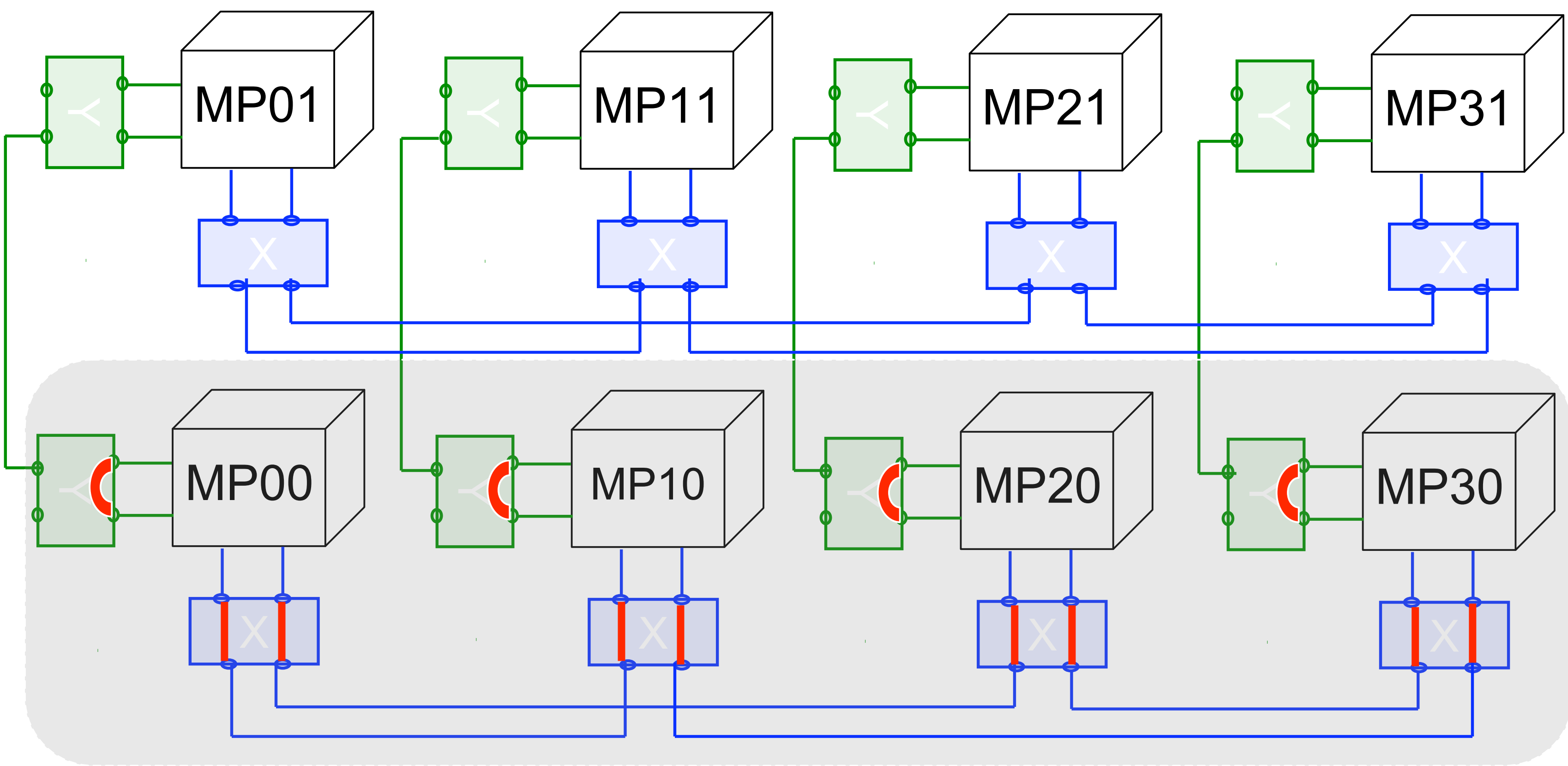


Partition Allocation on Multiple Midplanes – 1D Example

- Partitions are allocated in an isolated manner
  - No Congestion
  - Enhanced Security
- Partition can be connected as Torus or Mesh



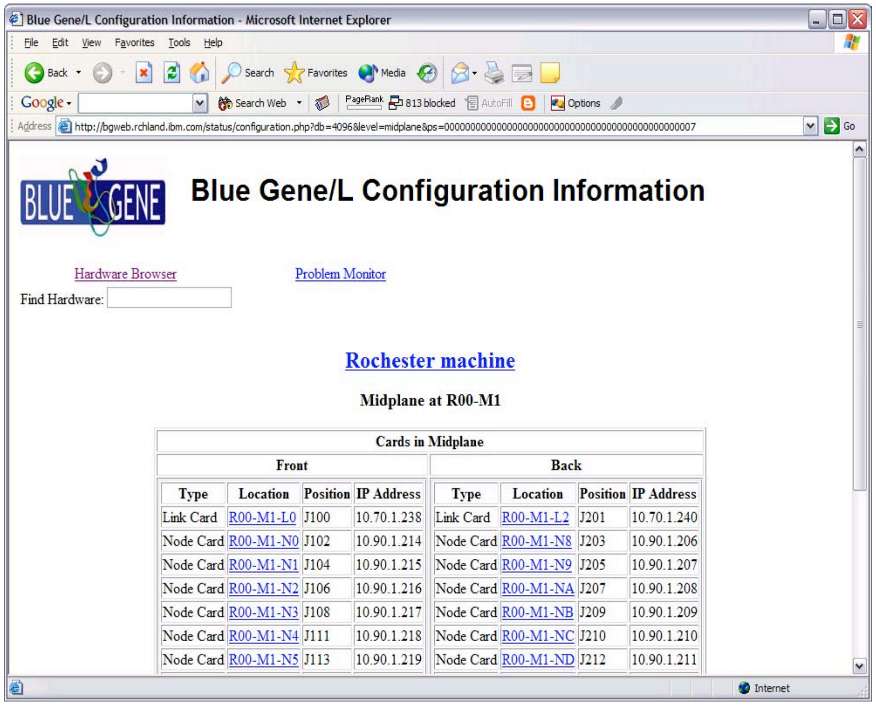
Example – A 2D Machine



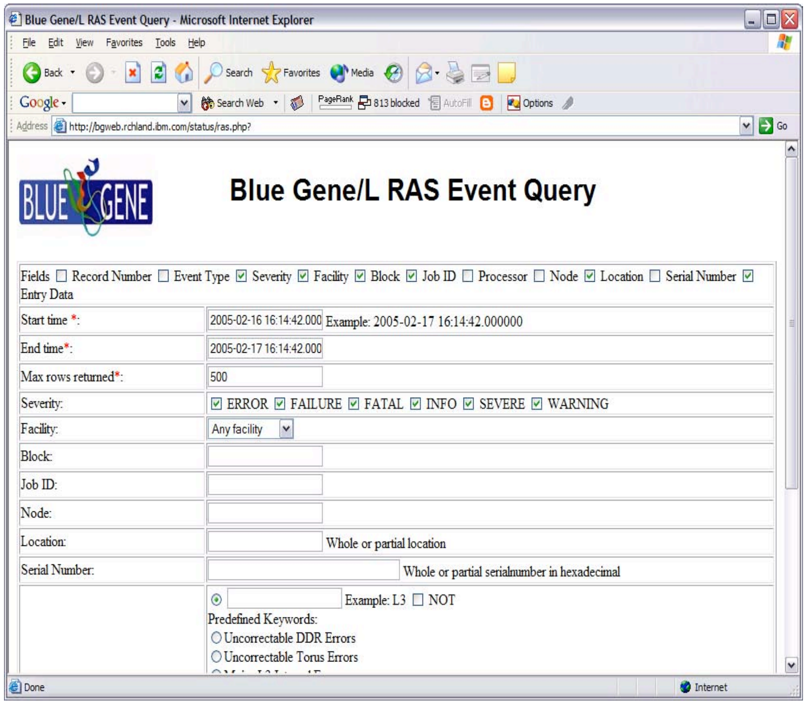
Web Interface to DB

- A front-end that runs via browser to view DB2 data.
- Supports the viewing of RAS data, configuration data, diagnostics data, and operational data.
- Can be used to see how the hardware fits together
- Can be used to find trouble areas, hardware anomalies
- Eliminates the need to have SQL expertise to view basic BlueGene information from the database.

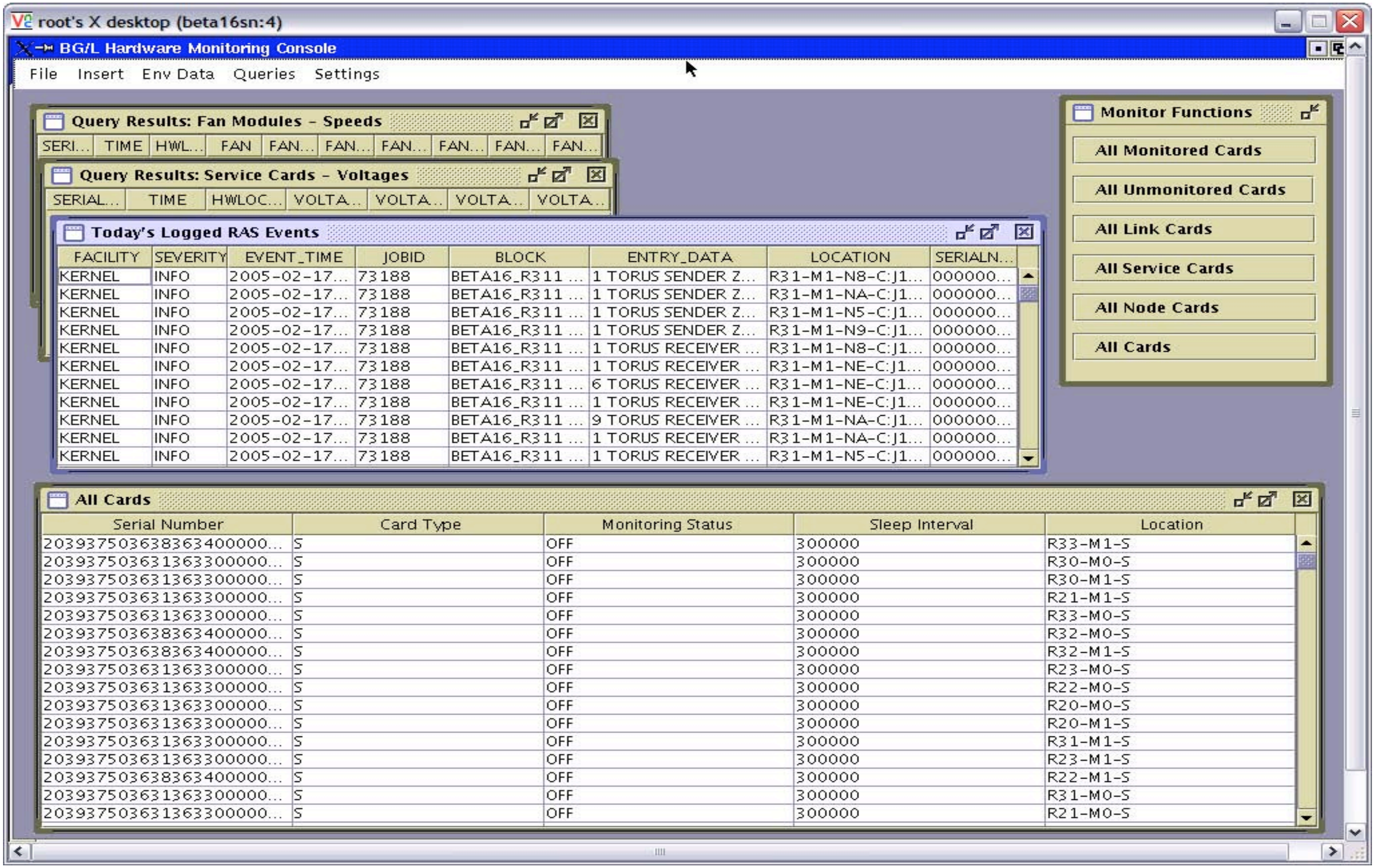
Web Page Screen Shots



Type	Location	Position	IP Address	Type	Location	Position	IP Address
Link Card	R00-M1-L0	J100	10.70.1.238	Link Card	R00-M1-L2	J201	10.70.1.240
Node Card	R00-M1-N0	J102	10.90.1.214	Node Card	R00-M1-N8	J203	10.90.1.206
Node Card	R00-M1-N1	J104	10.90.1.215	Node Card	R00-M1-N9	J205	10.90.1.207
Node Card	R00-M1-N2	J106	10.90.1.216	Node Card	R00-M1-NA	J207	10.90.1.208
Node Card	R00-M1-N3	J108	10.90.1.217	Node Card	R00-M1-NB	J209	10.90.1.209
Node Card	R00-M1-N4	J111	10.90.1.218	Node Card	R00-M1-NC	J210	10.90.1.210
Node Card	R00-M1-N5	J113	10.90.1.219	Node Card	R00-M1-ND	J212	10.90.1.211



Hardware Monitor Screen Shot



BG/L Control System Summary

- Control system software runs on the service node
- Control system components
  - MMCS
  - Discovery
  - Ciodb
  - Hardware Monitor
  - Proxy
  - RAS
- Control system handles:
  - Partitions
  - Jobs
  - Discovering and Monitoring Hardware
- DB2 is the central repository of all control system information

IBM Research

Parallel Filesystem

C. Howson

Feb. 24, 2005

Outline

- Parallel Filesystems on BlueGene/L
- Storage subsystems
- NFS
- GPFS
- ...
- Performance

Characteristics of Parallel Filesystems

- MPI-IO is most common client of parallel filesystem
  - Collective IO is very common, all nodes write to a different portion of same file
- Streaming IO all the way to disks is important
  - IO node ram is small, not much opportunity to cache
  - Aggregate ram is large, fileservers' cache may be too small as well
- Everything goes through GigE network in BlueGene/L
  - IO node is relatively underpowered, FS overhead will lower bandwidth

Storage Subsystems

- BlueGene/L is big, fileserver had better be able to support huge systems
  - Similar philosophy of many low cost disks
- Disk streaming performance is important
  - 15krpm U320 SCSI = ~20MB/s
  - 10krpm U320 SCSI = ~15MB/s
  - 7200rpm SATA = ???
- Metrics
  - Disks/U of rack space
  - Fileserver network bandwidth
    - IO nodes/Fileserver
    - Fileserver/Disks
- In many cases, fileserver is already there, BlueGene/L must support

Parallel Filesystems on BlueGene/L

- NFS
  - Simple, ubiquitous, relatively fast
  - Hybrid possible: BG/L nfs mounts from parallel fileserver
  - Poor support for shared file data between clients (MPI-IO)
- GPFS
  - Fully parallel filesystem: client writes directly to fileserver node with disk
  - Prototype runs on BlueGene/L, but needs tuning
- Others
  - Pvfs2
  - Lustre

GPFS Architecture

High capacity:

- Large number of disks in a single FS

High BW access to single file

- Large block size, full-stride I/O to RAID
- Wide striping – one file over all disks
- Multiple nodes read/write in parallel

High availability

- Nodes: log recovery restores consistency after a node failure
- Data: RAID or internal replication
- On-line management (add/remove disks or nodes without un-mounting)

Single-system image, standard POSIX interface

- Distributed locking for read/write semantics

GPFS Distributed Locking

- Distributed locking essential to …
  - synchronize file system operations for POSIX semantics,
  - synchronize updates to file system metadata on disk to prevent corruption,
  - maintain cache consistency of data and metadata cached on different nodes.
- Synchronization requires communication …
  - Problem: sending a lock message for every operation will not scale.
  - Solution: Token-based lock manager allows “lock caching”.

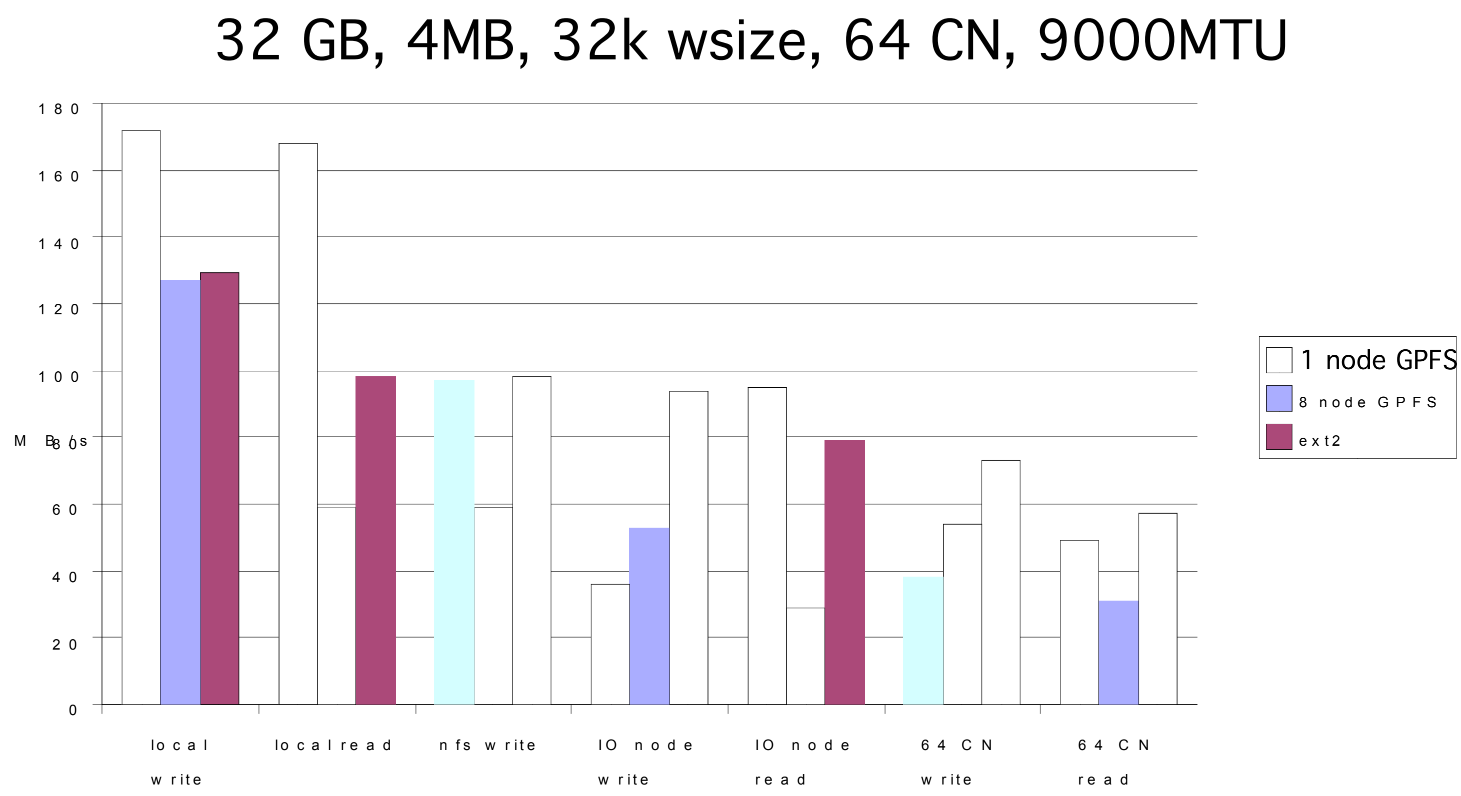
GPFS Token Based Locking

- Token server grants tokens.
- Token represents right to read, cache, and/or update a particular piece of data or metadata.
- Single message to token server allows repeated access to the same object.
- Conflicting operation on another node will revoke the token.
- Force-on-steal: dirty data & metadata flushed to disk when token is stolen.

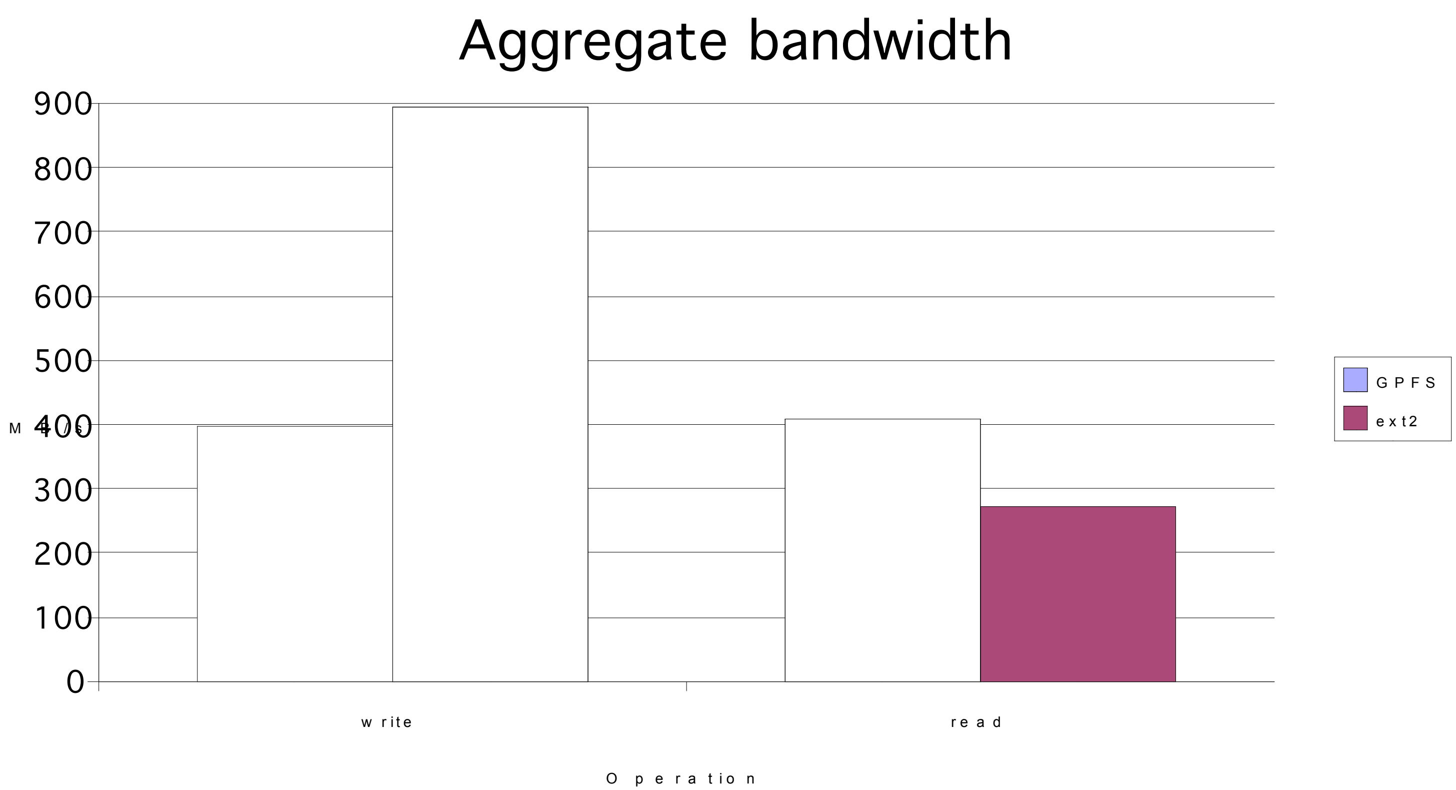
GPFS on BlueGene/L

- GPFS has client and manager nodes
  - Client only deals with own IO requirements, manager does token management as well
  - IO node is gpfs client
- GPFS consists of kernel module and user level daemon
  - kernel module handles local fs related functions
  - user level daemon handles external communications
- Initial prototype works
  - Performance is slow, due to debug build?
  - Need slight kernel modifications to support user level daemon

NFS client to GPFS or ext2 server



64 IO nodes NFS clients to 8 node GPFS or ext2 server



Conclusion

- BlueGene/L is high performance, so it needs a high performance filesystem
- Fileserver needs to scale up to large number of clients, servers, disks
- IO nodes don't have much RAM or computational power
- Tuning system parameters is very important, application dependent
  - NFS - rsize, wsize, tcp, udp, async, {r,w}mem_default,...
  - GPFS - pagepool, blocksize

IBM Research

BlueGene/L MPI From A User's Point Of View

- or -

A Short Survival Course In How To Annoy Tech Support

George Almási

Feb. 10, 2005

Outline

- This is not a talk about how to invoke mpirun.
  - System software folks will have presented that to you
- This is a talk about what does, and what doesn't work in BlueGene/L MPI.
  - Basic things you should avoid doing
  - Ideas about obtaining good performance
    - point-to-point messaging
      - scaling
      - mapping application into network
    - collective messaging

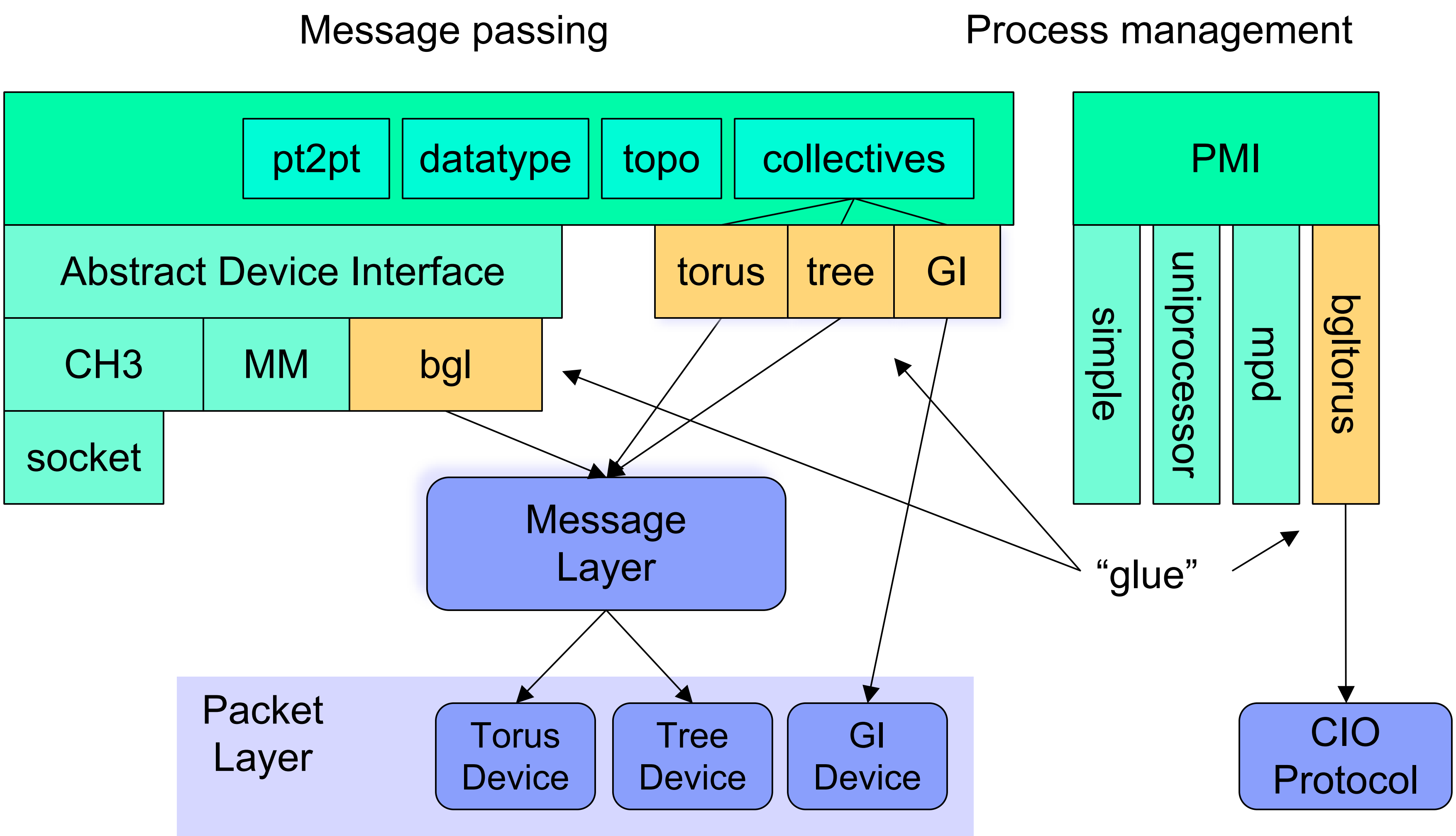
Summary I: will BlueGene like me?

- BlueGene/L MPI is like other implementations of MPI
  - looks like Argonne National Labs' MPICH2
  - because that's what it is.
- As an MPI developer you will have relatively few surprises
  - BlueGene/L MPI is MPI standard 1.2 compliant
    - no one-sided communication
    - no spawning of processes
    - supports thread model MPI_THREAD_SINGLE
    - MPI I/O is still under development
      - packages like HDF5, NETCDF are known to have been ported but presently display relatively poor performance
  - Getting higher performance out of BG/L MPI is inherently harder
    - large scale, fun network

Summary II: Kinds of annoyance you can cause

- **Crashing an application**: an opportunity to bad-mouth IBM
  - somewhat easier than on other platforms, because
    - limited memory on nodes, no virtual memory on nodes
      - memory leaks are going to make their presence felt
    - communication network is in userspace
      - good: high performance
      - bad: user have opportunity to kill process with wild pointers
- **Invoking the Halting Problem:** deadlocking the machine
- **Violating MPI semantics**: laws you didn't know were on the books
- **Causing bad performance:** malice not required
  - "nicely" map the application into the network (hard)
  - avoid load imbalance (hard)
  - avoid network jams (very hard)

BlueGene/L MPI Software Architecture



Write your own communication layer!

- 90% of communication hardware is mapped into user memory
  - Write stuff into high memory areas!
  - Likely to insert malformed packets into the torus.
    - will generate spurious error messages
    - will look somewhat like a system failure
    - may hang your application, and even bring down your partition.
  - Uninitialized pointer dereferences work great.
  - Requires very little effort to hang the machine
    - If you know what you are doing.
  - We have never seen this happen by accident.
    - You cannot accidentally malloc() over network hardware
    - You must set your pointer into a narrow address space

Deadlocking the system

- Talk before you listen.
- Illegal MPI code
  - find it in most MPI books
  - tech support will be very annoyed
- BlueGene/L MPI is designed not to deadlock easily.
  - It will likely survive this code.
- This code will cause MPI to allocate memory to deal with unexpected messages. If MPI runs out of memory, it will stop with an error message

```
CPU1 code:

MPI_Send (cpu2);
MPI_Recv(cpu2);

CPU2 code:

MPI_Send(cpu1);
MPI_Recv(cpu1);
```

Force MPI to allocate too much memory

- Post receives in one order, sends in the opposite order
- This is legal MPI code
- BlueGene/L MPI will choke if the sum of buffers is greater than the amount of physical memory
  - this is an implementation defect that will be fixed in the future

CPU1 code:

MPI_ISend(cpu2, tag_1);

MPI_ISend(cpu2, tag_2);

...

MPI_ISend(cpu2, tag_n);

CPU2 code:

MPI_Recv(cpu1, tag_n);

MPI_Recv(cpu1, tag_{n-1});

...

MPI_Recv(cpu1, tag_1);

Sneaky: violate MPI buffer ownership rules

- write send/receive buffers before completion
  - results in data race on any machine
- touch send buffers before message completion
  - not legal by standard
  - BG/L MPI will survive it today
  - no guarantee about tomorrow
- touch receive buffers before completion
  - BG/L MPI will yield wrong results

```
req = MPI_Isend (buffer);
buffer[0] = something;
MPI_Wait(req);
```

```
req = MPI_Isend (buffer);
z = buffer[0];
MPI_Wait (req);
```

```
req = MPI_Irecv (buffer);
z = buffer[0];
MPI_Wait (req);
```

Causing memory overruns: never wait for MPI_Test

- Have to wait for all requests
  - The standard requires waiting
  - or testing until MPI_Test returns true
- This code works on many other architectures
  - causes tiny memory leaks
- On BG/L this will run the system out of memory very fast
  - MPI_Request requires a lot of memory
  - It's a scaling issue

```
req = MPI_Isend( ... );
MPI_Test (req);
... do something else; forget about
   req ...
```

Straddle collectives with point-to-point messages

- On the ragged edge of legality
- BlueGene/L MPI works
- Multiple networks issue:
  - Isend handled by torus network
  - Barrier handled by GI network

```
CPU 1 code:

req = MPI_Isend (cpu2);
MPI_Barrier();
MPI_Wait(req);
```

```
CPU 2 code:

MPI_Recv (cpu1);
MPI_Barrier();
```

Send flood

- This is legal MPI code
  - also ... stupid MPI code
  - not scalable, even when it works
- BlueGene/L MPI will run out of buffer space
  - This is a bug, and will be fixed
- We have seen this kind of code in the wild
  - Don't write code such as this
  - Even if you think it should work

CPU 1 to n-1 code:

```
MPI_Send(cpu0);
```

CPU 0 code:

```
for (i=1; i<n; i++)
    MPI_Recv(cpu[i]);
```

BlueGene/L ≠ Linux

- You are likely to run into surprises with what you assume runs on the compute nodes
  - Don't try asynchronous File I/O
  - TCP client stuff works:
    - socket(), connect()
  - TCP server stuff doesn't work:
    - bind(), accept()
  - BG/L runs sleep(10000) in 6 seconds!

Virtual Node Mode vs. Coprocessor mode

- Virtual Node Mode:
  - twice the processing power!
  - but not twice the performance
    - half of memory per CPU
    - half of cache per CPU
    - half of network per CPU
    - CPU has to do both computation and communication
- Coprocessor mode:
  - only one CPU available to execute user code
  - but have all memory!
  - other CPU helps with communication
  - currently, only point-to-point communication benefits
    - that is about to change

Point-to-point performance (I)

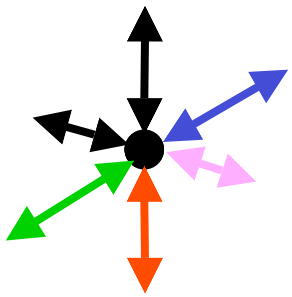
- Two kinds of network routing on BlueGene/L
  - deterministic routing:
    - each packet goes along the same path
    - maintains packet order
    - creates network hotspots
  - adaptive routing
    - packets overtake
    - equalized network load
    - harder on CPUs
    - MPI matching semantics are always correct!

- MPI Short protocol:
  - very short (<250 bytes) messages. Deterministically routed
- MPI Eager protocol:
  - medium size messages
  - “send without asking”
  - deterministically routed
  - latency around 3.3 μs
- MPI Rendezvous protocol:
  - large (> 10KBytes) messages
  - adaptively routed
  - bandwidth optimized

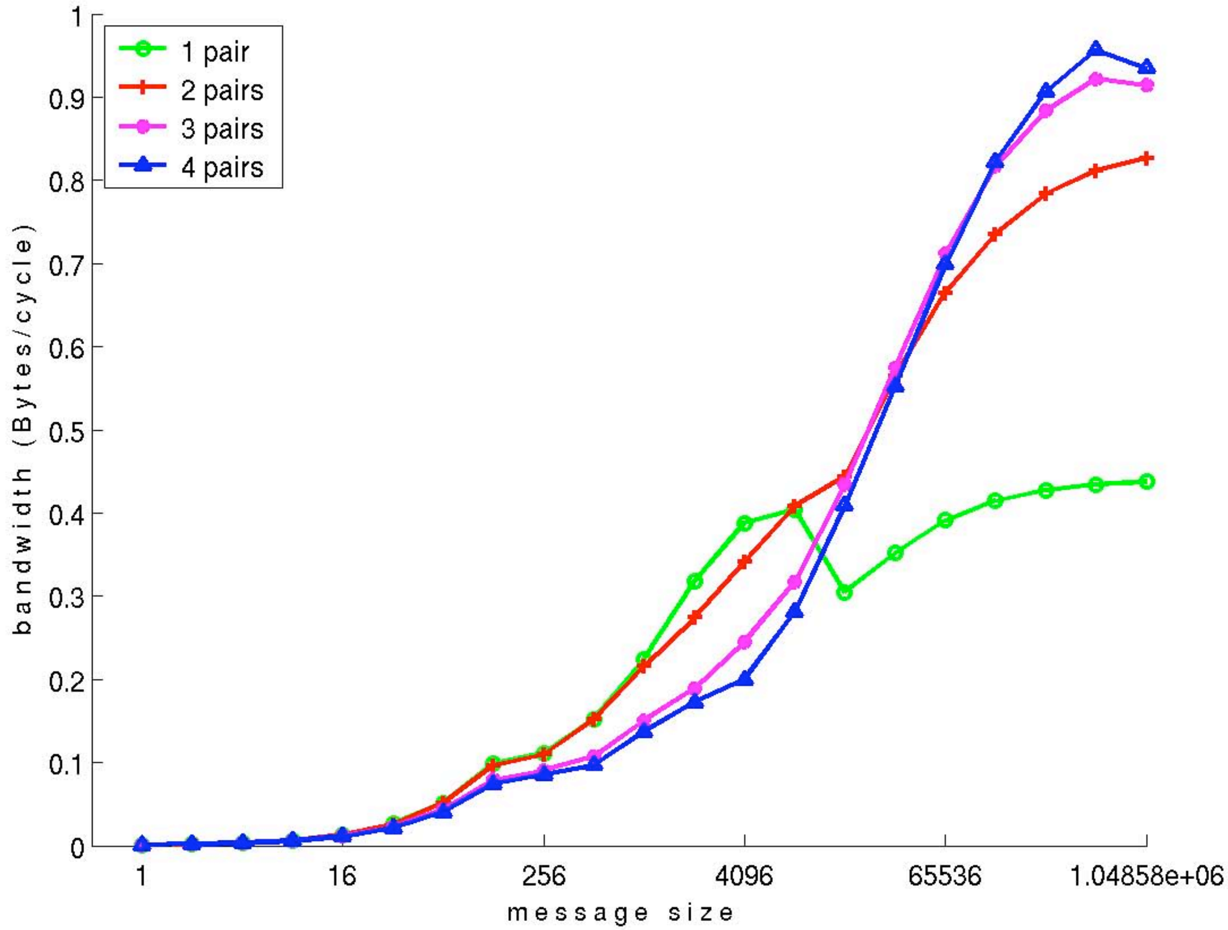
Point to point performance (II)

- The rendezvous treshold (10KBytes) can be changed
  - environment variable: BLMPI_RZV = ...
- Lower the rendezvous treshold if
  - running on a large partition
  - many short messages are overloading the network
  - eager messages are creating artificial hotspots
  - program is not latency sensitive
- Increase the rendezvous treshold if
  - most communication is nearest-neighbor
    - or at least close in Manhattan distance
  - relatively longer messages
  - you need better latency on medium size messages

Bandwidth vs. message size



6-way send+recv



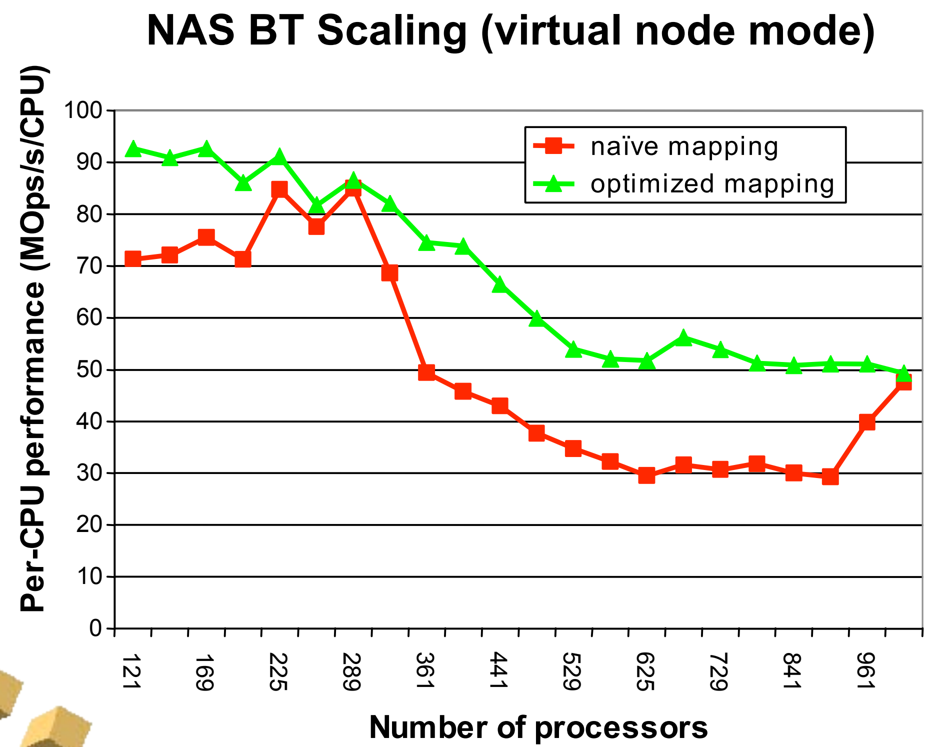
1 byte/cycle = 700MB/s

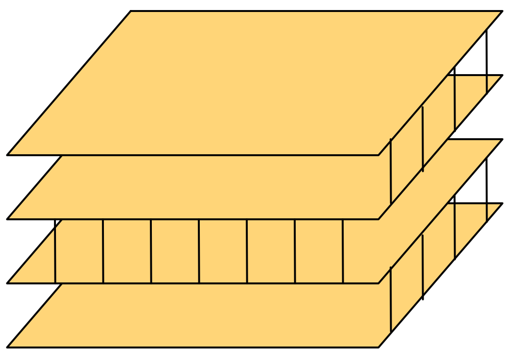
Point-to-point performance (III): Dos and Don'ts

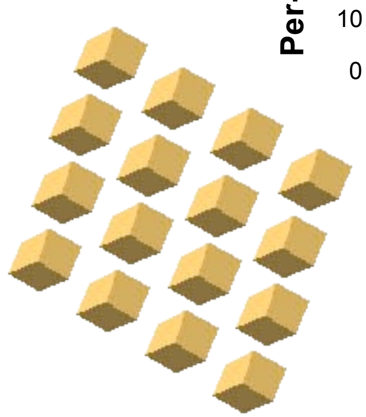
- Overlapping communication and computation:
  - requires care on BlueGene/L
  - keep programs in sync as much as you can
    - alternate computation and communication phases
- Avoid load imbalance
  - bad for scaling
- Shorten Manhattan distance messages have to traverse
  - send to nearest neighbors!
- Avoid synchronous sends
  - increases latency
- Avoid buffered sends
  - memory copies are bad for your health
- Avoid vector data, non-contiguous data types
  - BG/L MPI doesn't have a nice way to deal with them
- Post receives in advance
  - unexpected messages damage performance

The all-important torus mapping

- NAS BT
  - 2D mesh communication pattern
  - Map on 3D mesh/torus?
    - Folding and inverting planes in the 3D mesh
- NAS BT scaling:
  - Computation scales down with n^{-2}
  - Communication scales down with n^{-1}







How to map an application to the torus?

- set up a mapping file

```
0 0 0 0
1 0 0 0
2 0 0 0
3 0 0 0
...
```

- associate torus coordinates to MPI ranks 0 to n-1.
- Yeah, but why quadruplets?
- Use mapping file as argument in mpirun invocation

MPI Collective performance (I)

- Rule 1: Use collectives whenever you can
  - Point-to-point performance has huge overheads
  - "I can do a better job with point-to-point than you miserable jocks can do with collectives"
    - We don't think so.
- Rule 2: Mapping is all-important for good collective performance
  - Most collective implementations prefer certain communicator shapes
- Rule 3: don't do anything crazy, like
  - Use different buffer sizes for a broadcast call (illegal)
  - Use heterogeneous data types for broadcast (legal, but crazy)
  - Use misaligned buffers (legal and not crazy, but we don't like it anyway)
  - Run point-to-point messages across the communicator at the same time that a collective is underway (legal, but not cheap)

Summary of Optimized BG/L MPI Collectives

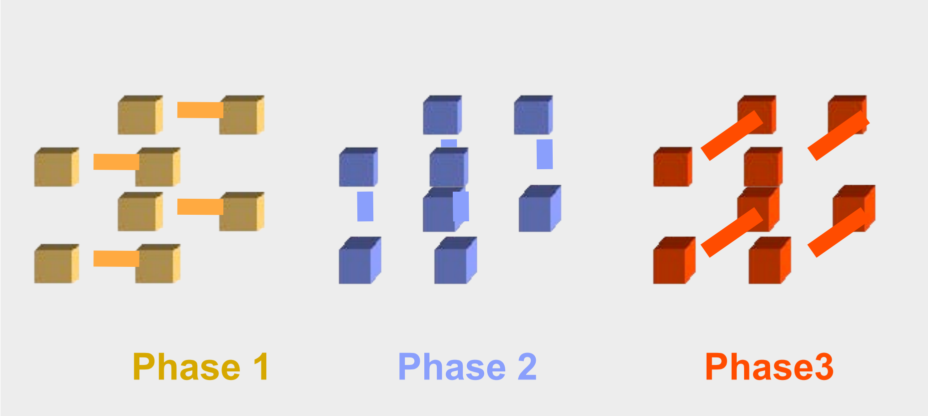
	Condition	Network	Performance
Barrier	COMM_WORLD	GI	1.5us
	COMM_WORLD	Tree	5 us
	Rectangular communicator	Torus	10-15 us
Broadcast	COMM_WORLD	Tree	350 Mbytes/s
	Rectangular communicator	Torus	320 Mbytes/s (0.48 Bytes/cycle)
	Rectangular communicator	Torus	**TBD**: low latency
Allreduce	COMM_WORLD, fixed point	Tree	350 Mbytes/s, low latency
	COMM_WORLD, floating pt	Tree	40 Mbytes/s (0.06Bytes/c)
		Tree	**TBD:** >= 120MB/s, low latency
	Hamilton Path	Torus	120 Mbytes/s
	Rectangular communicator	Torus	80 Mbytes/s
	Rect. comm. + short msg	Torus	10-15 us latency
	TBD: other shapes	Torus	**TBD:** high bandwidth FP
Alltoall[v]	Any communicator	Torus	84-97% of peak
Allgatherv	rectangular	Torus	Same as broadcast

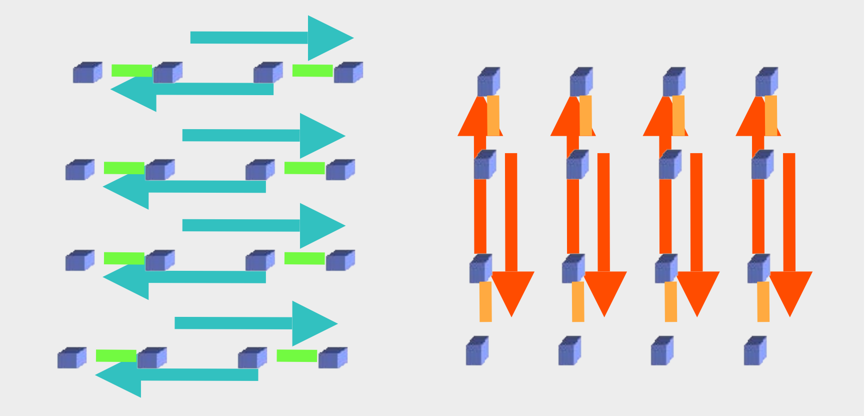
Optimizing collective performance: Barrier and short-message Allreduce

- **Barrier** is implemented as an **allgather** in each dimension
  - BG/L torus hardware can send deposit packets on a line
  - Low latency broadcast
- Since packets are short, likelihood of conflicts is low
- Latency = O(xsize+ysize+zsize)

- **Allreduce** for very short messages is implemented with a similar multi-phase algorithm

impl. by Yili Zheng

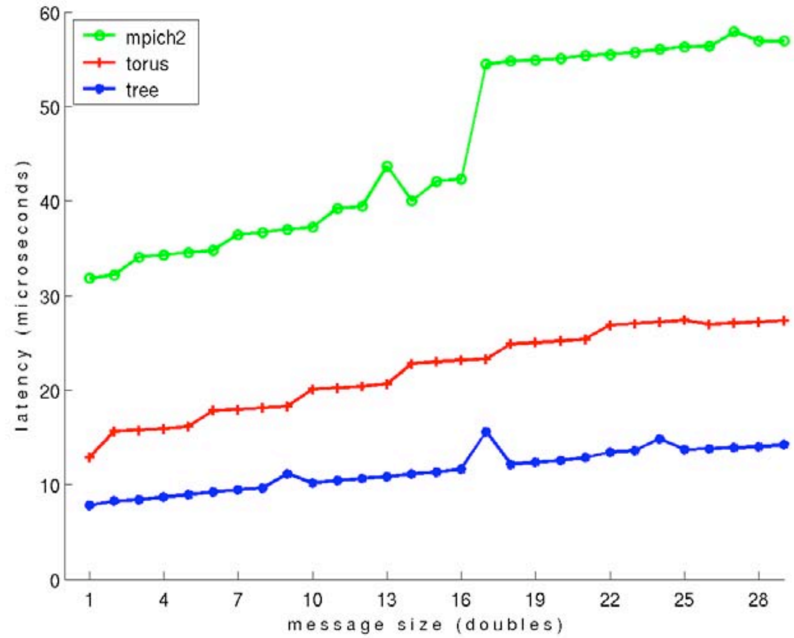




Barrier and short message Allreduce: Latency and Scaling

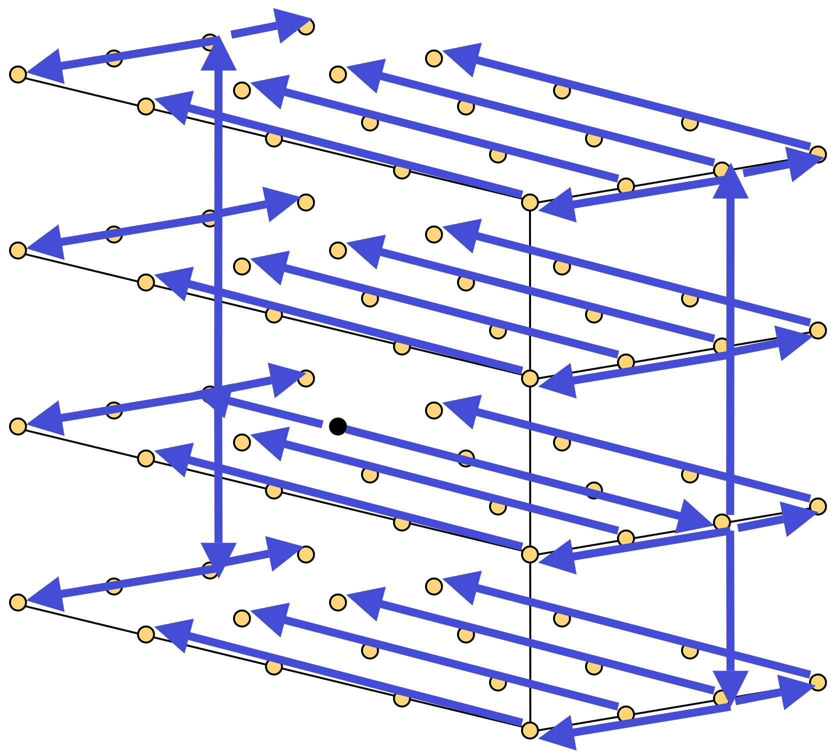
Barrier latency vs. machine size

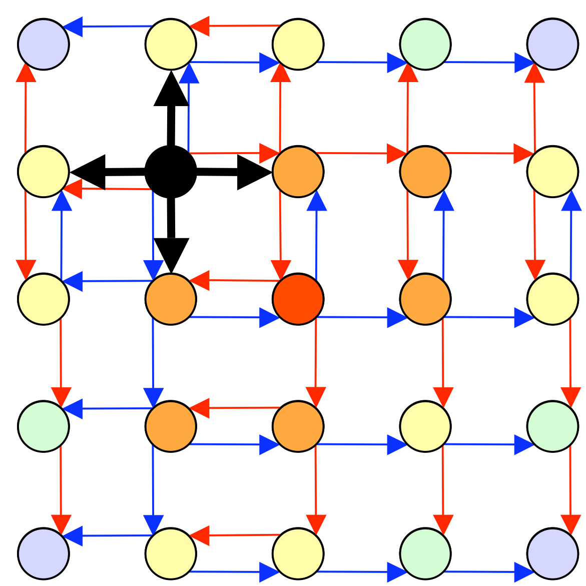
Short-message Allreduce latency vs. message size



MPI_Bcast on a mesh: algorithm details

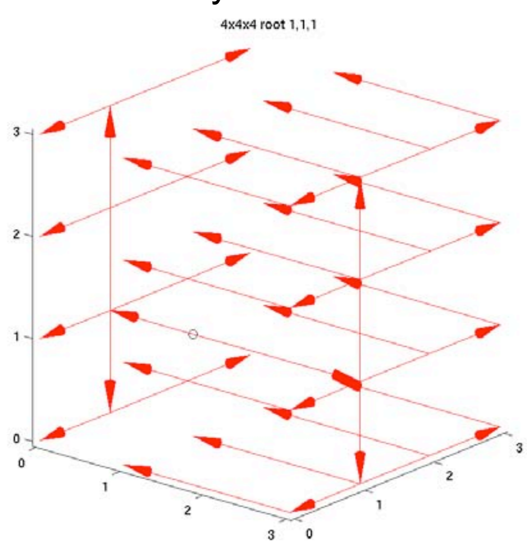
with John Gunnels, Nils Smeds, Vernon Austel, Yili Zheng, Xavier Martorell

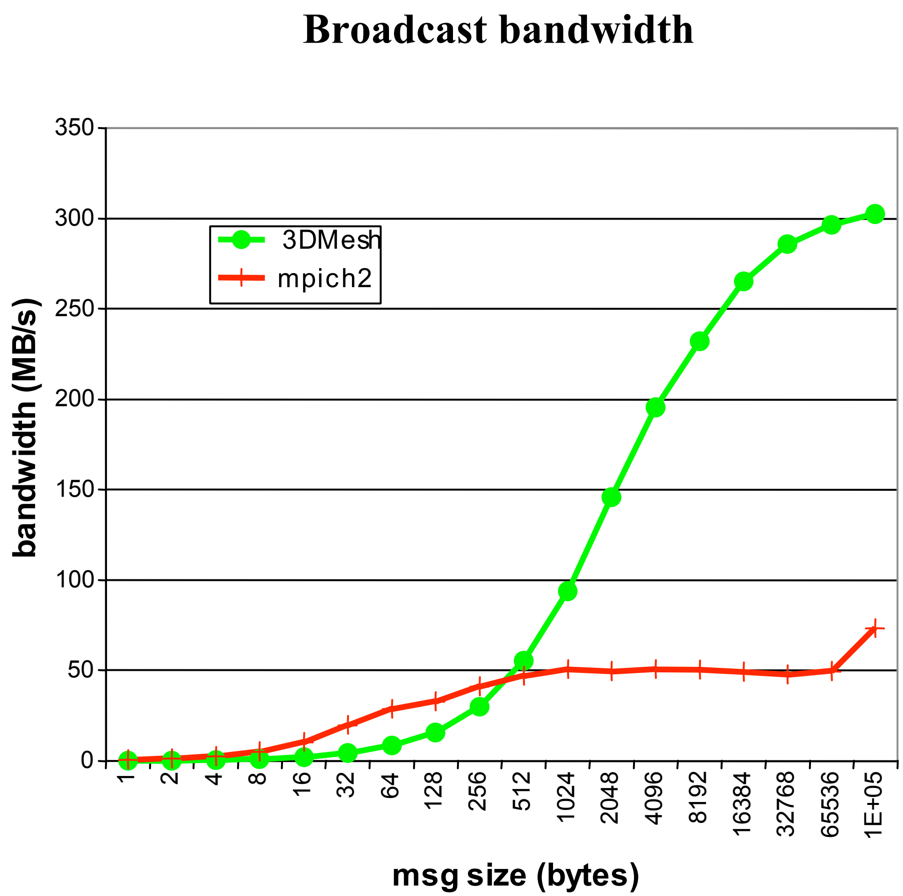




MPI_Bcast performance

- MPICH2: stable but slow
- Tree broadcast:
  - only for MPI_COMM_WORLD
- Torus broadcast:
  - any rectangular communicator
  - Uses deposit bit
  - “menu” system

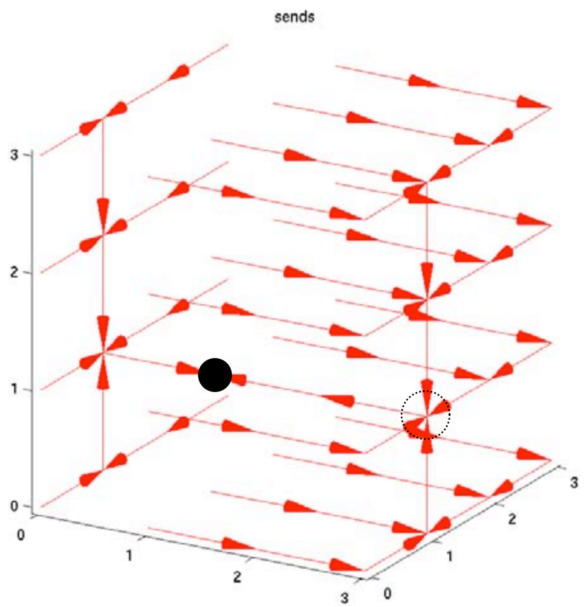


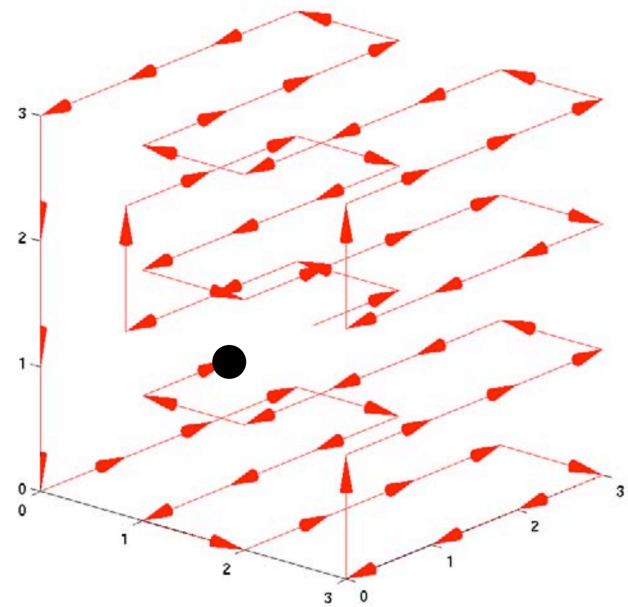


Optimized collectives: Allreduce for long messages

- Allreduce: standard "menu"
  - Similar to broadcast
  - Reasonable latency
  - Strongly CPU limited

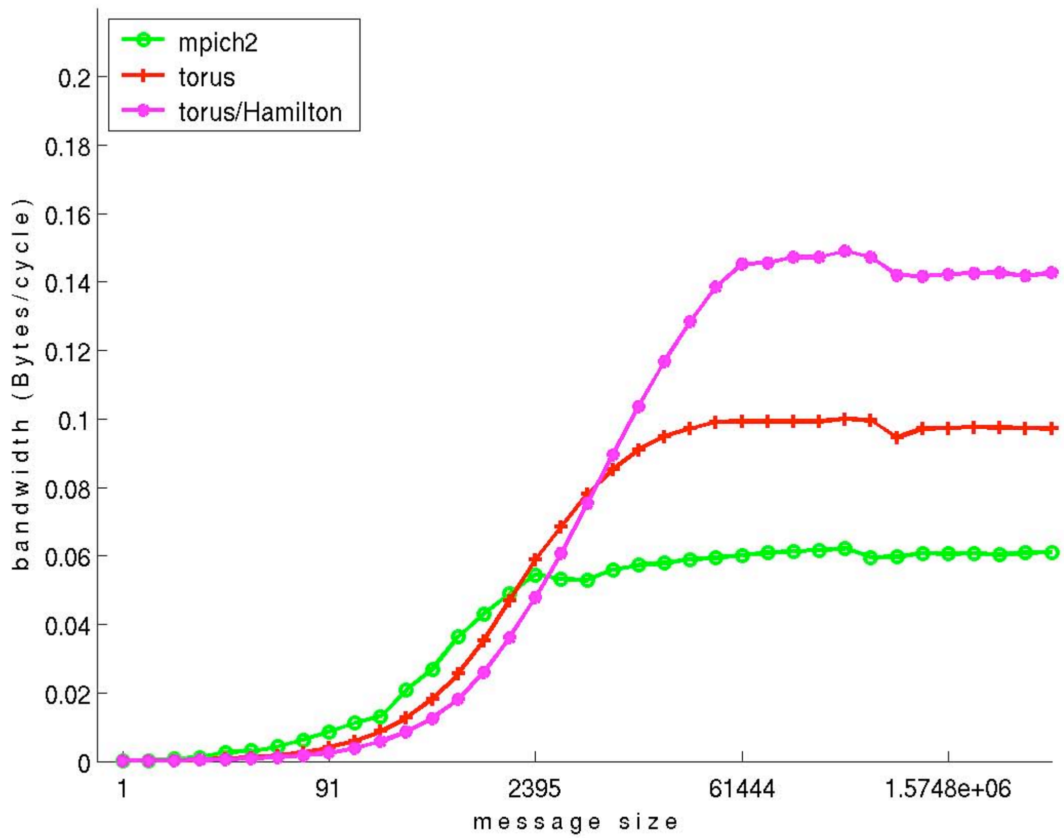
- Allreduce: Hamiltonian path "menu"
  - Single line snaking through torus
  - Very high latency
  - Somewhat better bandwidth





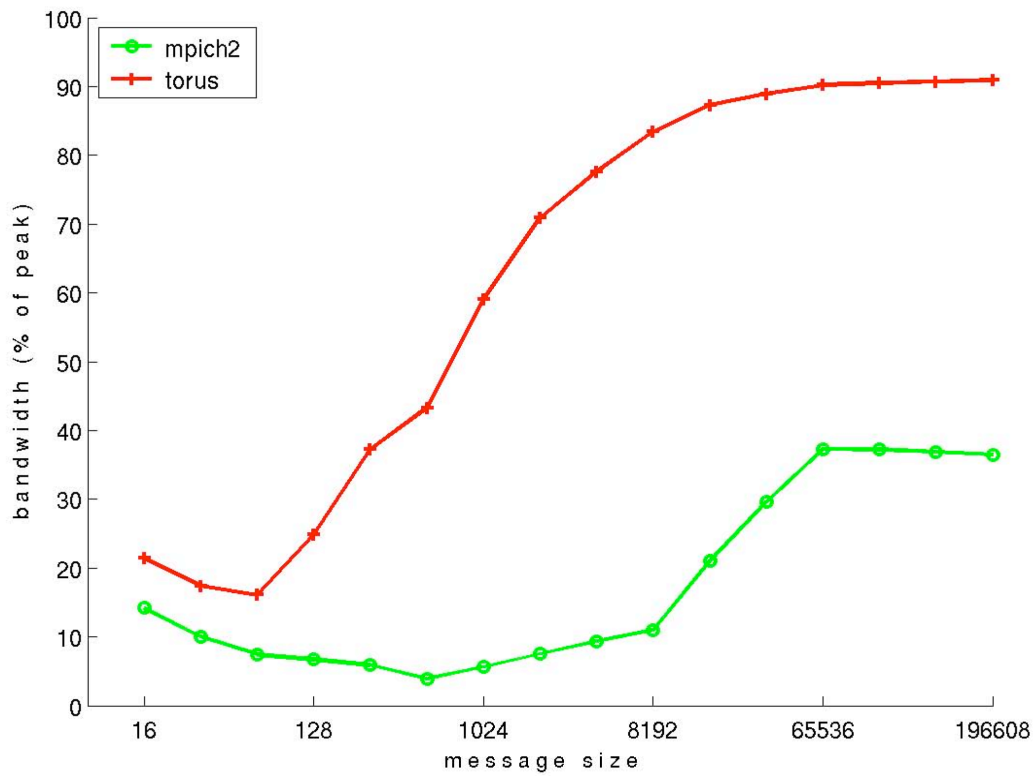
Impl. by Chris Erway

Optimized collectives: Allreduce bandwidth



Optimized collectives: Alltoall[v]

- Performance measured as percentage of peak, which is function of partition “shape”
- MPICH2 implementation not suitable for torus network
- Optimized implementation: 90% of peak
- **Impl. by Charles Archer**
- measured on an 8x8x8 partition



Conclusion

- You have been warned.
  - If you call tech support you will get asked tedious questions about the things I have outlined in this presentation.
- BG/L MPI is a moving target. Some things are going to improve over the next few months
  - flow control to handle send flood issues
  - better optimized collective performance
  - MPI I/O
- We would love to hear about your porting experience.

MPI on BG/L at ANL

Rusty Lusk
Mathematics and Computer Science Division
Argonne National Laboratory

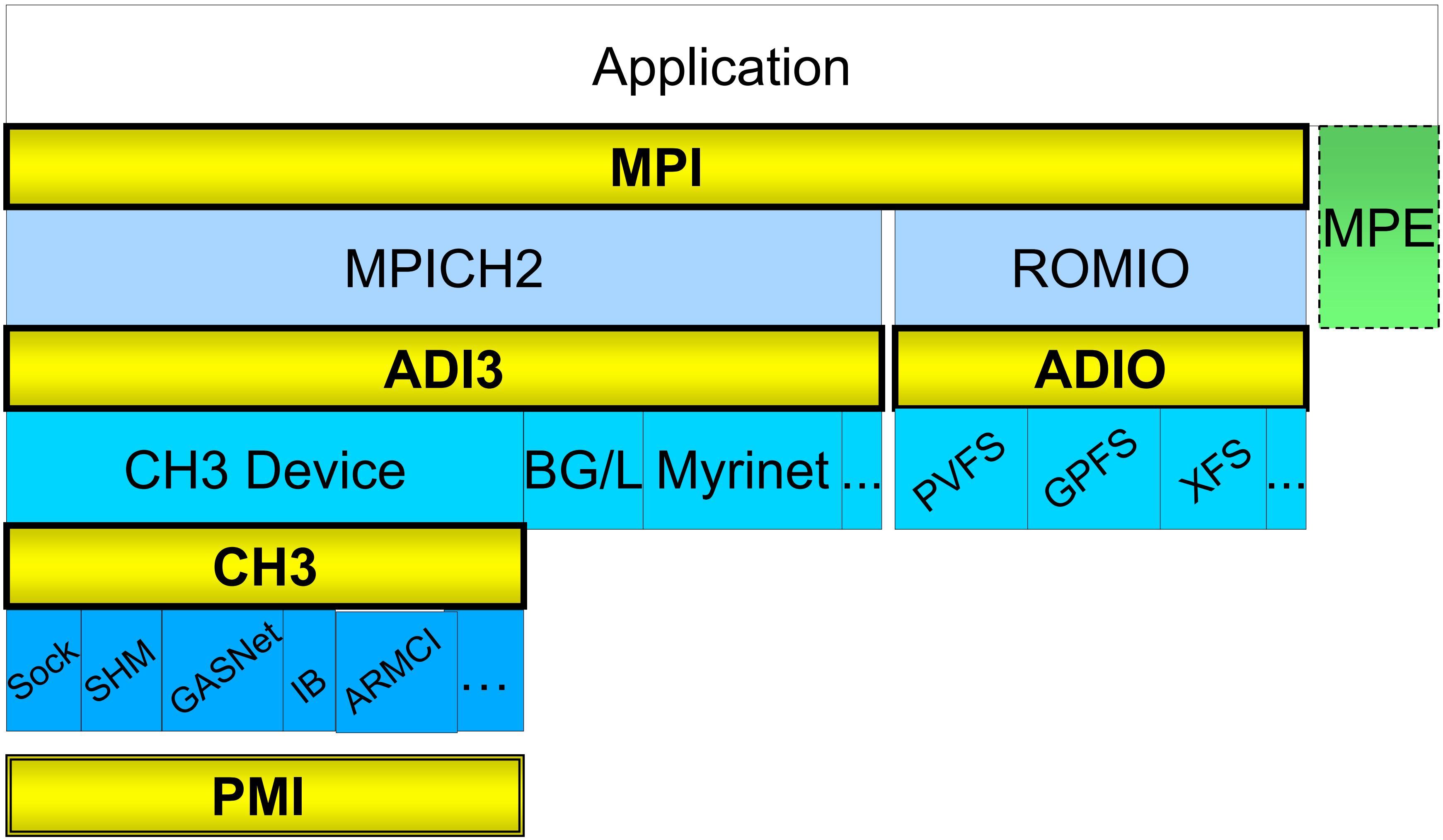
Outline

- MPI, MPI-2, and MPICH2
- Some basic experimental results on MPI performance
- A short look at a performance tool on BG/L
- Some near-term relevant ANL projects
- Some longer-term potential collaborative projects

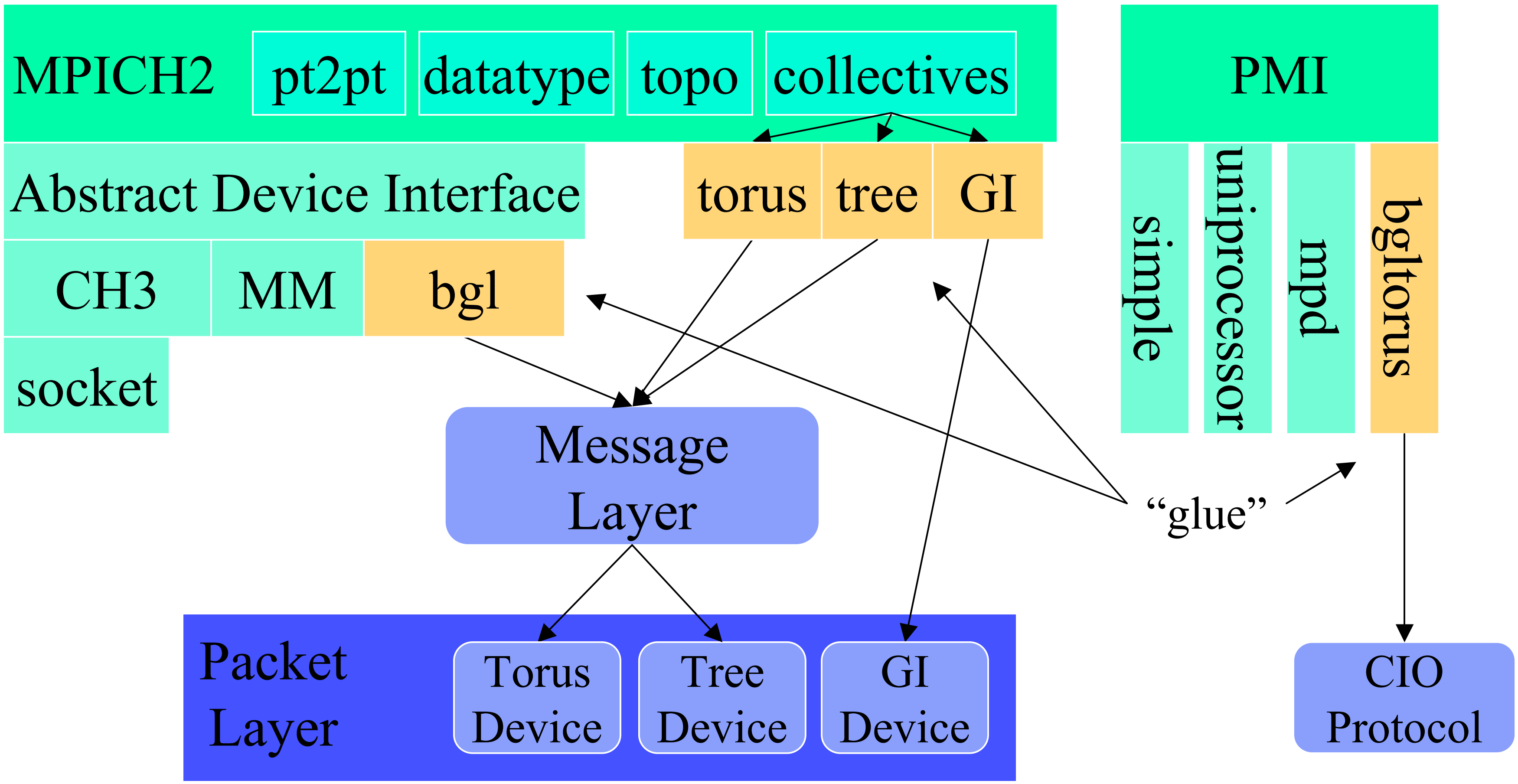
MPI Implementation at ANL and IBM

- MPICH2 is an all-new, open-source, portable implementation of the full MPI-2 standard
- Several vendors are using it as the basis of their MPI implementations.
- IBM and ANL have collaborated on using MPICH2 as the basis of BG/L's MPI (MPI-1, so far)

MPICH2 Structure



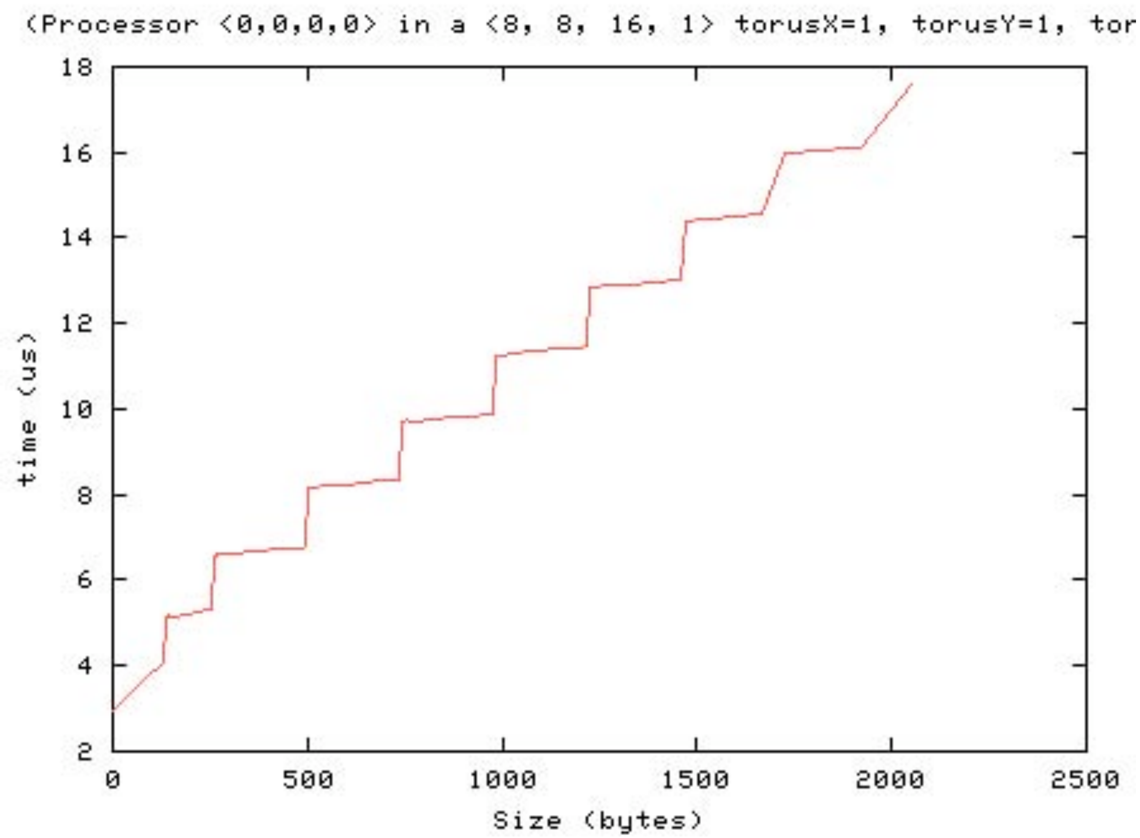
IBM BG/L MPI Software Architecture



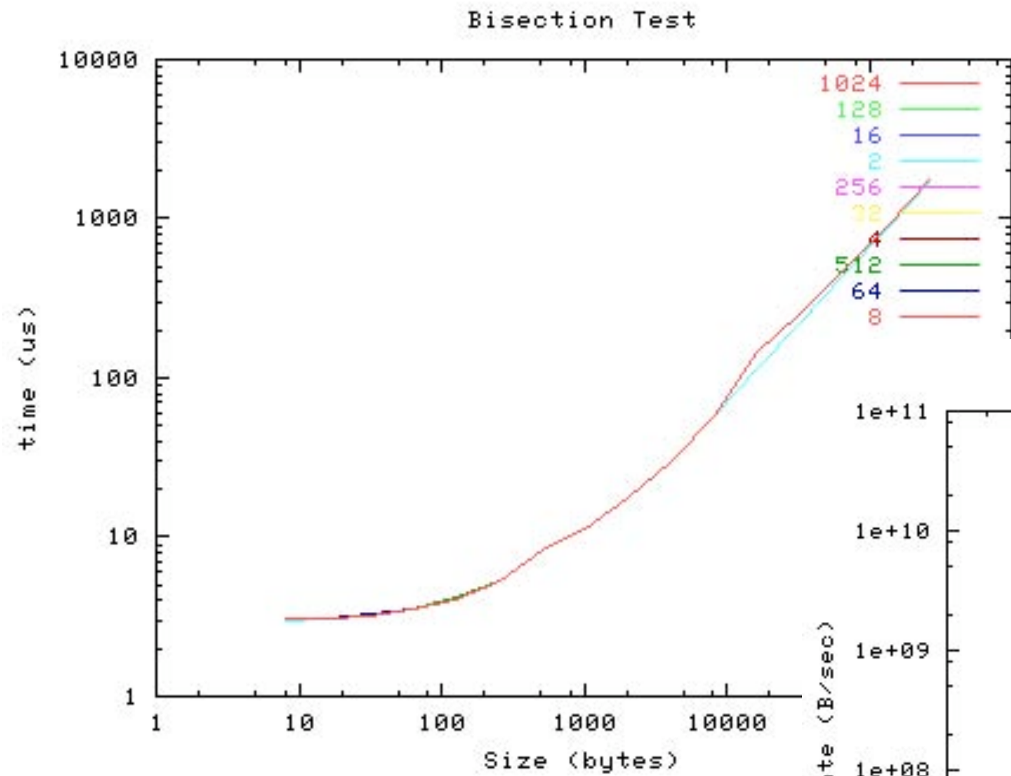
This slide courtesy of IBM

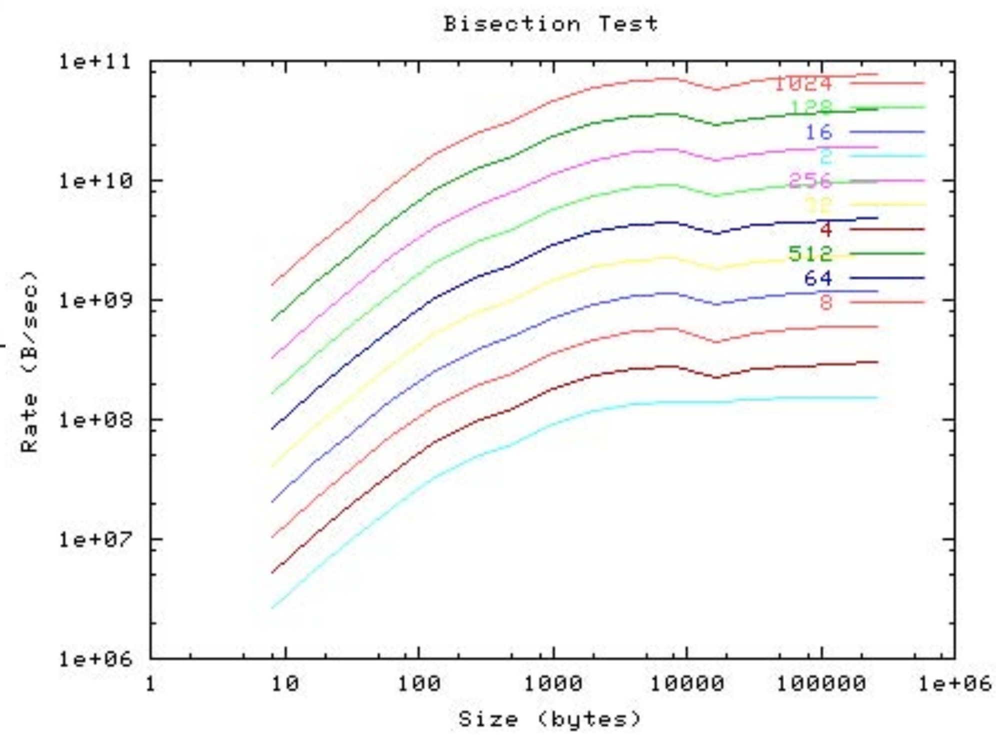
Performance Summary

- Recent tests on Argonne's one-rack machine
- From http://www.mcs.anlo.gov/~gropp/projects/parallel/BGL/mpptest (other tests nearby)
- Latency:

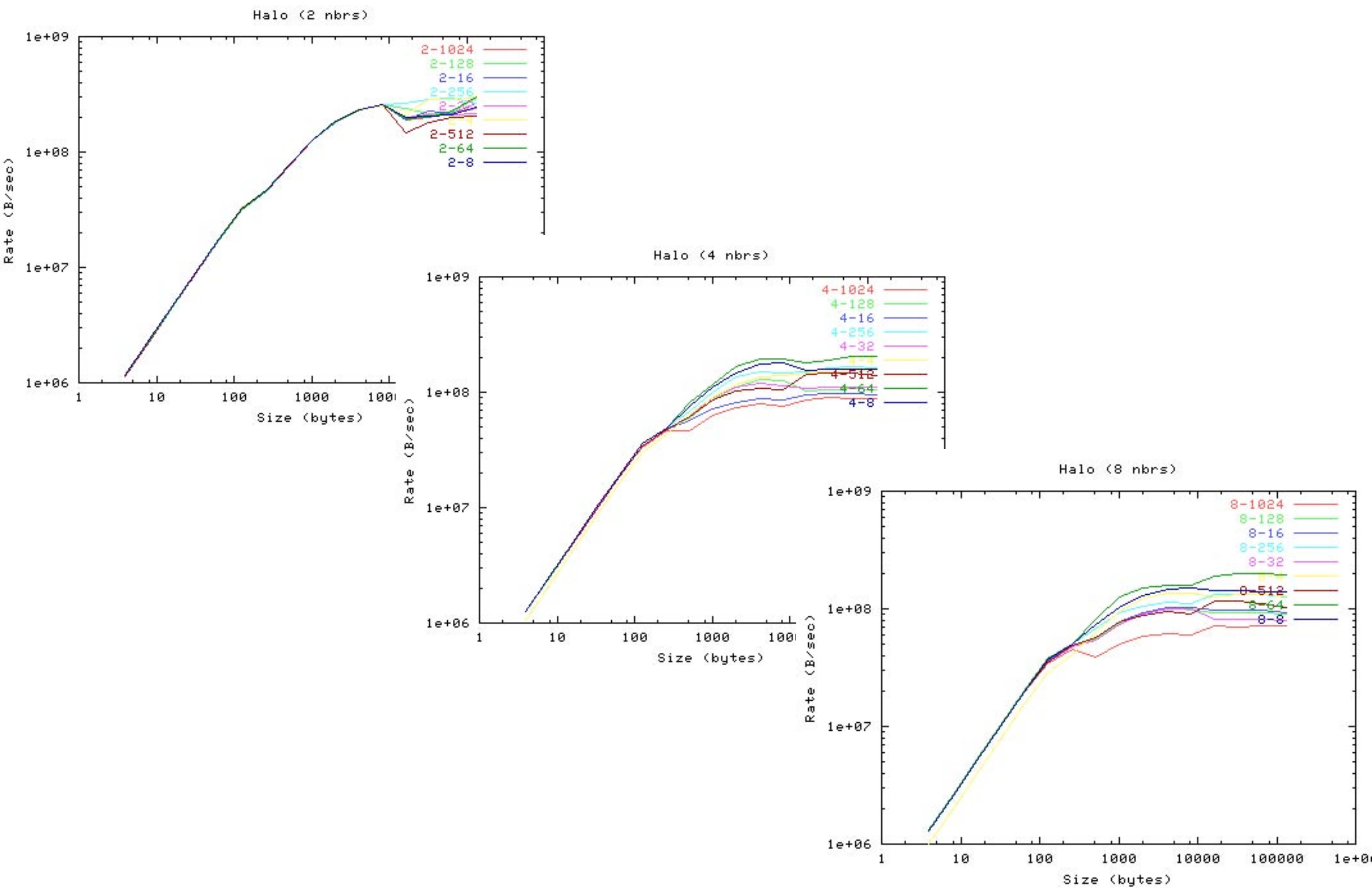


Bisection Performance

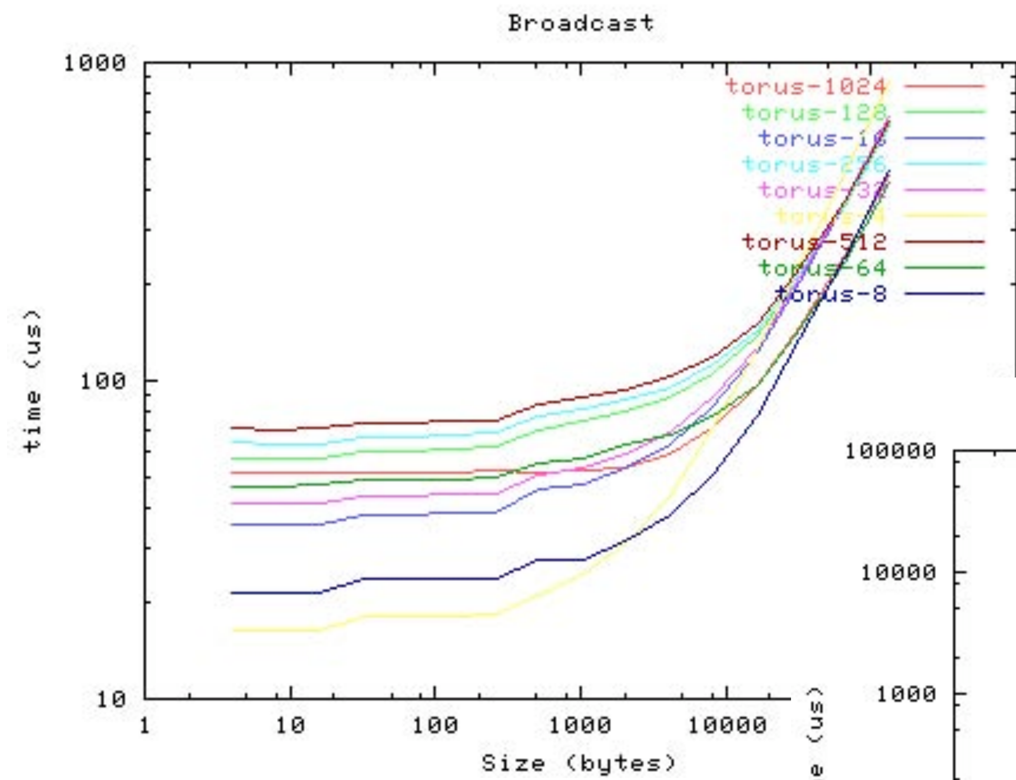


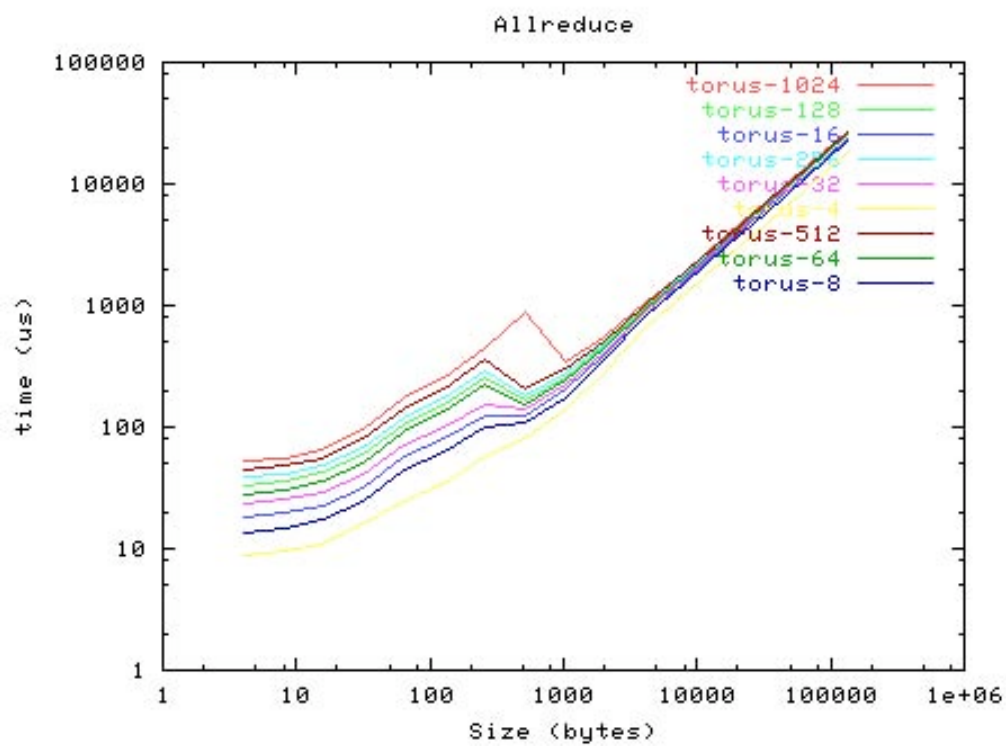


Halo Exchange



Collective Operations



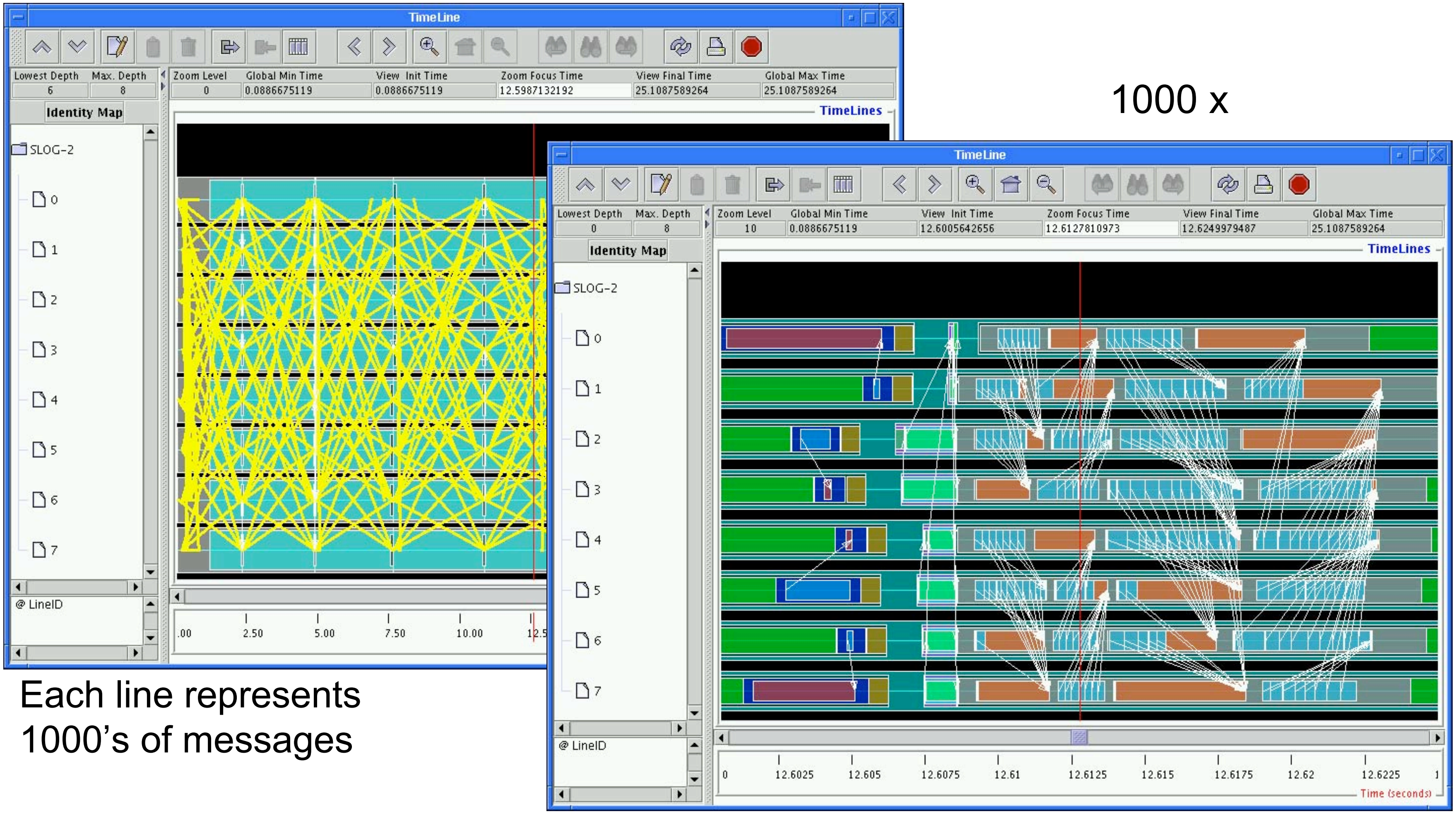


MPI Performance Summary

- With our own tests on our own machine, BG/L’s MPI is reliable, fast, consistent, and scalable.

Jumpshot on BG/L

1000 x

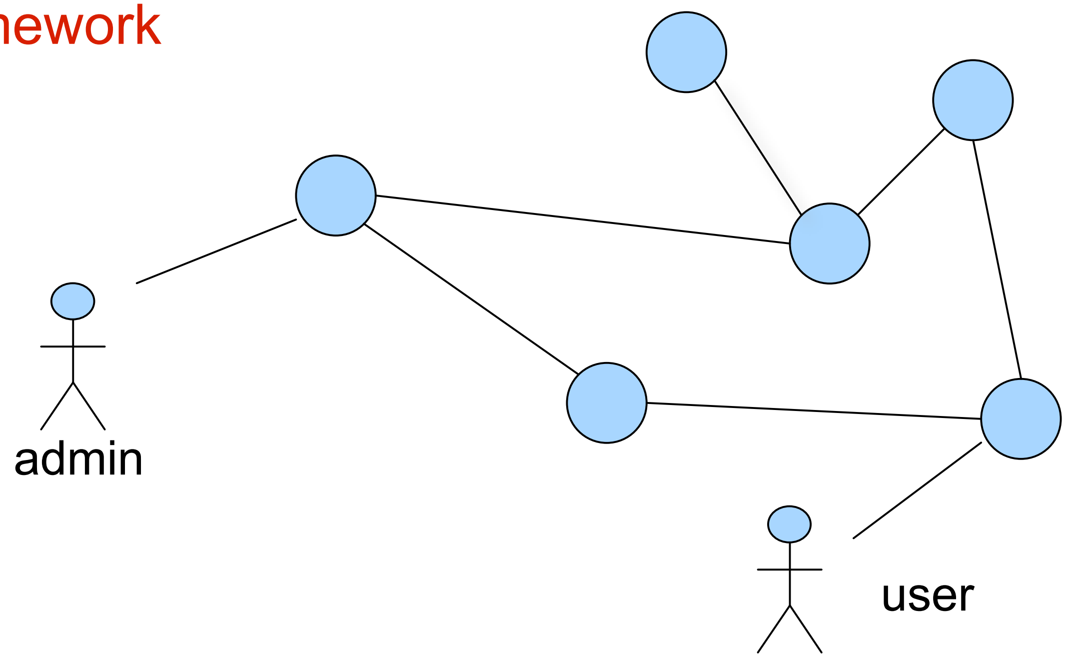


Each line represents 1000's of messages

Detailed view shows opportunities for optimization

The System Software Environment

- We are participants in the Scalable Systems Software SciDAC project, which has developed a component architecture for system software
- Whole suite currently running on Chiba City cluster
- Some components being ported to BG/L environment
  - Communication framework
  - Process manager
  - Queue manager
  - Scheduler



Near-Term MPI Projects

- MPICH2 release 1.0.1 (this week) has slots for specialized topology routines, analogous to slots for specialized collective routines. We plan to work with IBM to ensure that this approach enables new optimizations.
  - Should help applications tune themselves
- We plan to conduct research and develop new MPICH collective routines that base their algorithms on the topology routines.
- Incorporating new optimized datatype handling
- MPICH2 supports the MPI standard **`mpiexec`** with several useful extensions (such as passing environment variables). We hope to merge our approach with IBM's **`mpirun`**, which is a work in progress.
- Porting some system software components
  - Integrating with existing IBM system software

Longer-term Collaborative MPI Projects

- MPI-2 (already in MPICH2)
  - MPI-I/O to a fast parallel file system
    - Rob’s talk: need better language for implementing ROMIO on compute nodes
  - One-sided operations
    - We currently are working with a neuroscience application that is a good match to MPI_Put/Get
    - Short-term: port MPICH2 version
    - Long-term: customized for BG/L
- System software
  - Several ways to improve the environment seen by users being explored

The End

PLAN GLIMB

Rusty Lusk

Mathematics and Computer Science Division

Argonne National Laboratory

IBM Research

On Developing BlueGene/L MPI-IO with High Performance

Hao Yu (yuh@us.ibm.com)

Feb. 23, 2005

On developing BG/L MPI-IO with high performance

- What is MPI-IO and BG/L MPI-IO?
- Status of BG/L MPI-IO
  - Functionalities
  - A preliminary performance
- On-going efforts
- Summary

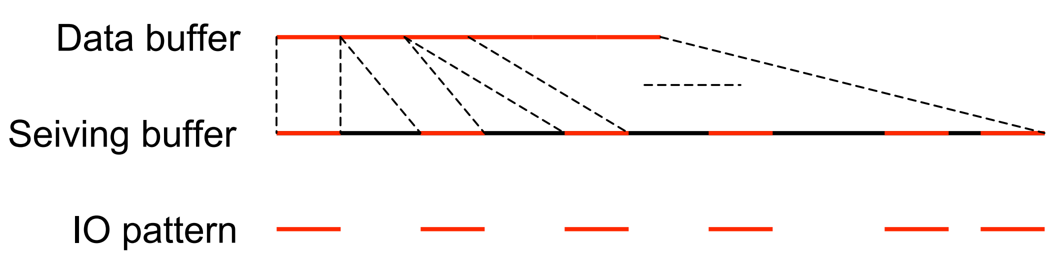
What is MPI-IO

- Parallel I/O interface specified in MPI-2 standard
  - Supports portable high performance file IO
  - File view functions and MPI datatypes allow user to express complex IO patterns
  - 3 orthogonal aspects of data access functions
    - File positioning: explicit offset, individual file pointer, shared file ptr;
    - Synchronism: blocking, non-blocking;
    - Coordination: non-collective (independent), collective
    - Among these, collective file accesses allow MPI-IO to optimize the interactions with storage devices.
  - File consistency: atomic/non-atomic access mode
  - File manipulation: open, pre-allocate, resize, etc.

Example #1: non-contiguous IO

```
int blocksize[4] = {2,2,2,2};
int indices[4] = {0,3,9,18};
char buf[8];

MPI_Type_indexed( 4, blocksize, indices, MPI_BYTE, filetype )
MPI_Type_commit( &filetype );

MPI_File_open(… &fhandle);
MPI_File_set_view( fhandle,offset,MPI_BYTE, filetype, "native", info);

MPI_File_read( fhandle, buf, 8, MPI_BYTE, &status );
```

MPI-IO may optimize the non-contiguous read by data sieving or using GPFS prefetch hints.



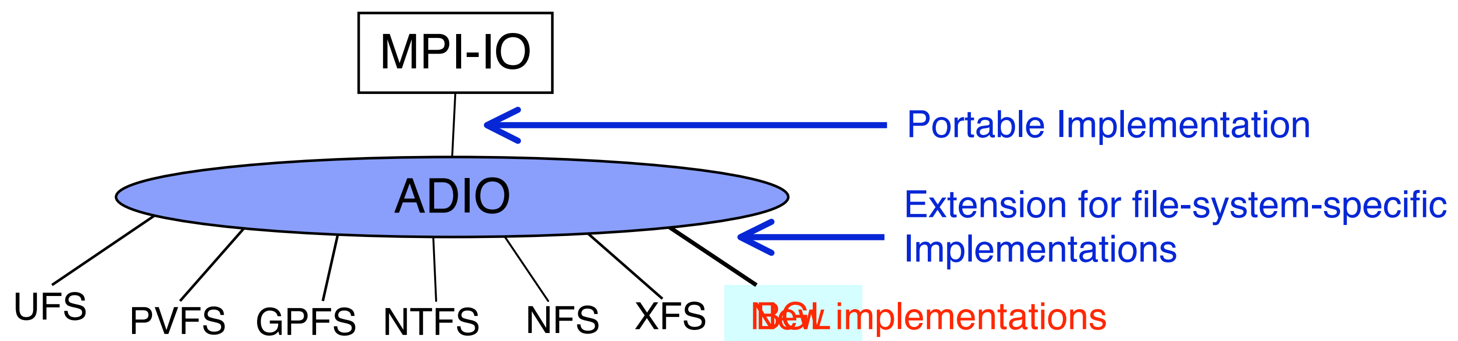
Example #2: collective non-contiguous IO

```
int blocksize[4] = {2,2,2,2};
char buf[8];

if      (myrank == 0) indices[4] = {0,4,8,12};
else if (myrank == 1) indices[4] = {2,6,10,14};

MPI_Type_indexed( n, blocksize, indices, MPI_BYTE, filetype )
MPI_Type_commit( &filetype );

MPI_File_open(… &fhandle);
MPI_File_set_view( fhandle,offset,MPI_BYTE, filetype, "native", info);

/* read from 4 disjoint regions from file */
MPI_File_read_all( fhandle, buf, 8, MPI_BYTE, &status );
```

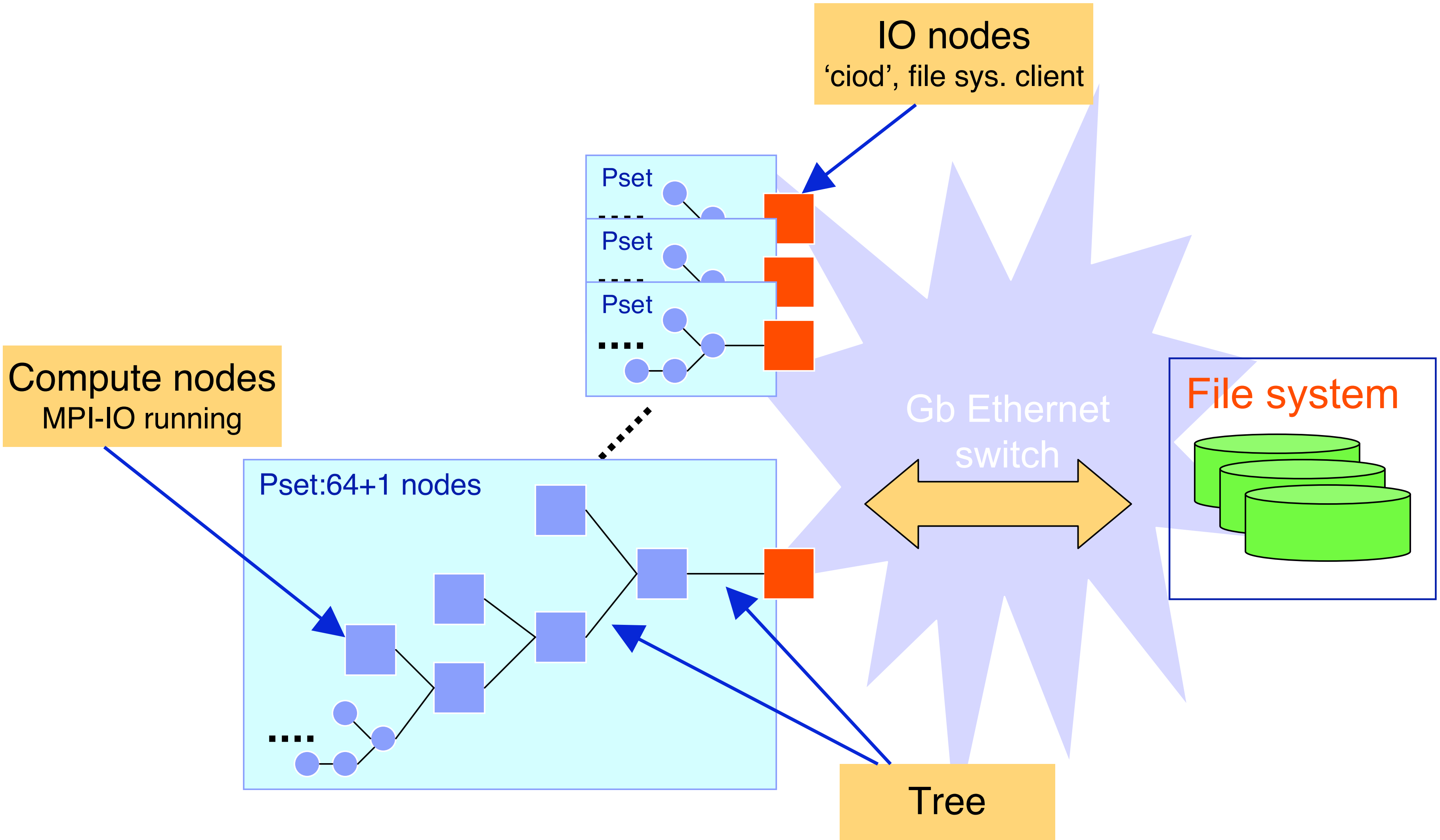
MPI-IO may aggregate the read requests from 2 processes and issues contiguous IO operations to the file system.

What is BlueGene/L MPI-IO

- BlueGene/L MPI-IO started as a direct port of Argonne National Lab's MPI-IO implementation, ROMIO.
- What is ROMIO
  - Aportable MPI-IO implementation
  - Its portability is achieved mainly because that it was built on top of MPI and an abstract-device interface called ADIO
  - Emphasizing on optimizing collective IO and non-contiguous IO
- BG/L MPI-IO took ROMIO implementation for NFS



BG/L I/O subsystem

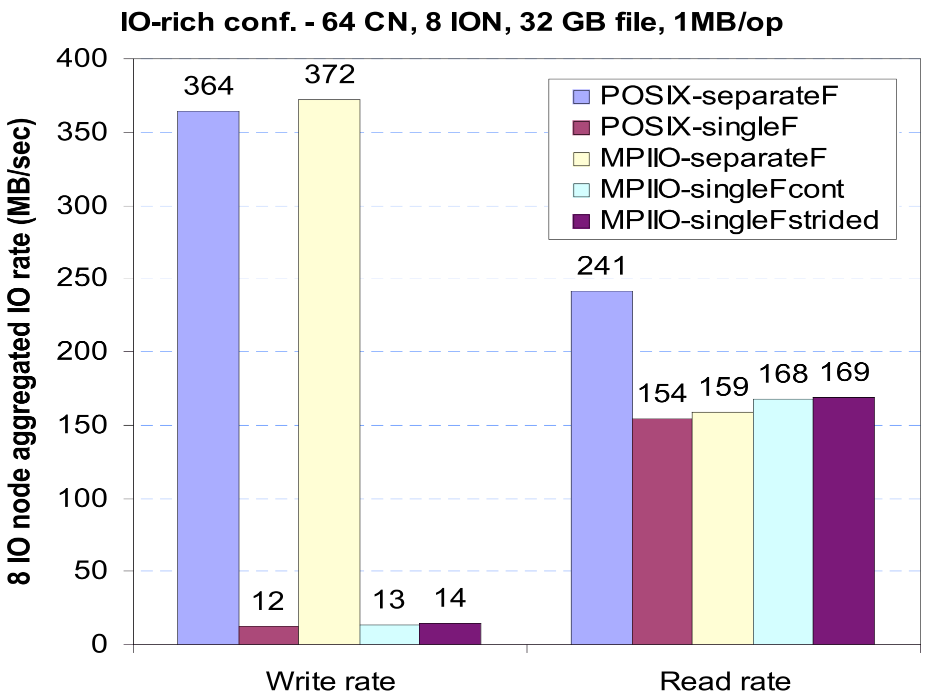


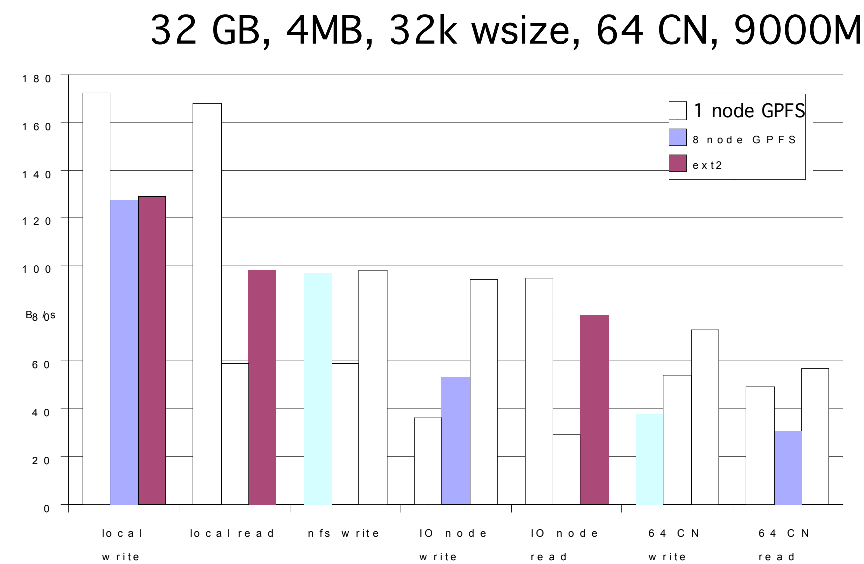
Status of BG/L MPI-IO

- Ported most MPI-IO functionalities
  - MPI-IO functionalities are tested for ROMIO tests, MPICH2 IO tests, LLNL MPIO-test, parallel HDF5, PnetCDF. FLASH_bench
  - Exchanged many emails with ROMIO team at ANL.
  - Enhanced BG/L with fcntl() file locking function (can be easily extended for supporting MPI-IO atomic access mode for other file systems)
- Started work on performance optimization for BG/L MPI-IO
  - Optimization for collective IO
  - GPFS specific developments
  - Collaborations:
    - ANL ROMIO team (optimization for PVFS2)
    - Northwestern U: Choudhary, Coloma, Ching

Preliminary MPI-IO performance…

IOR 2.8.1, 8 IO nodes, NFS mount, 8-node GPFS, 1.7TB

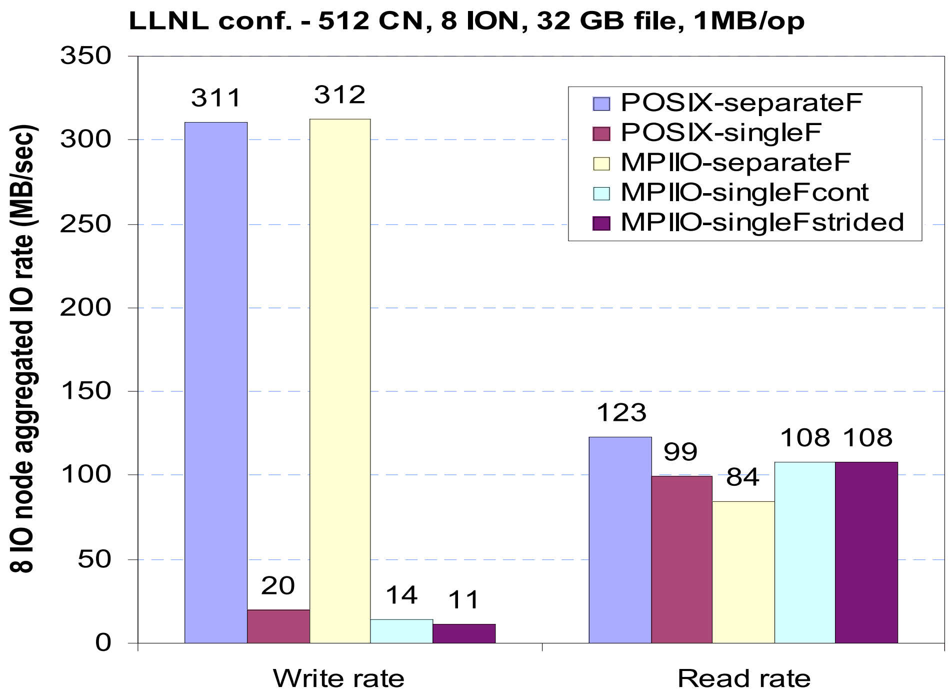




- Reason for the poor single file writing performance: we did not specify “noac” for NFS mount. Every wsize NFS write invokes metadata lock.
- Reason for the 160MBps read performance is not clear.

… Preliminary MPI-IO performance

IOR 2.8.1, 8 IO nodes, NFS mount, 8-node GPFS, 1.7TB

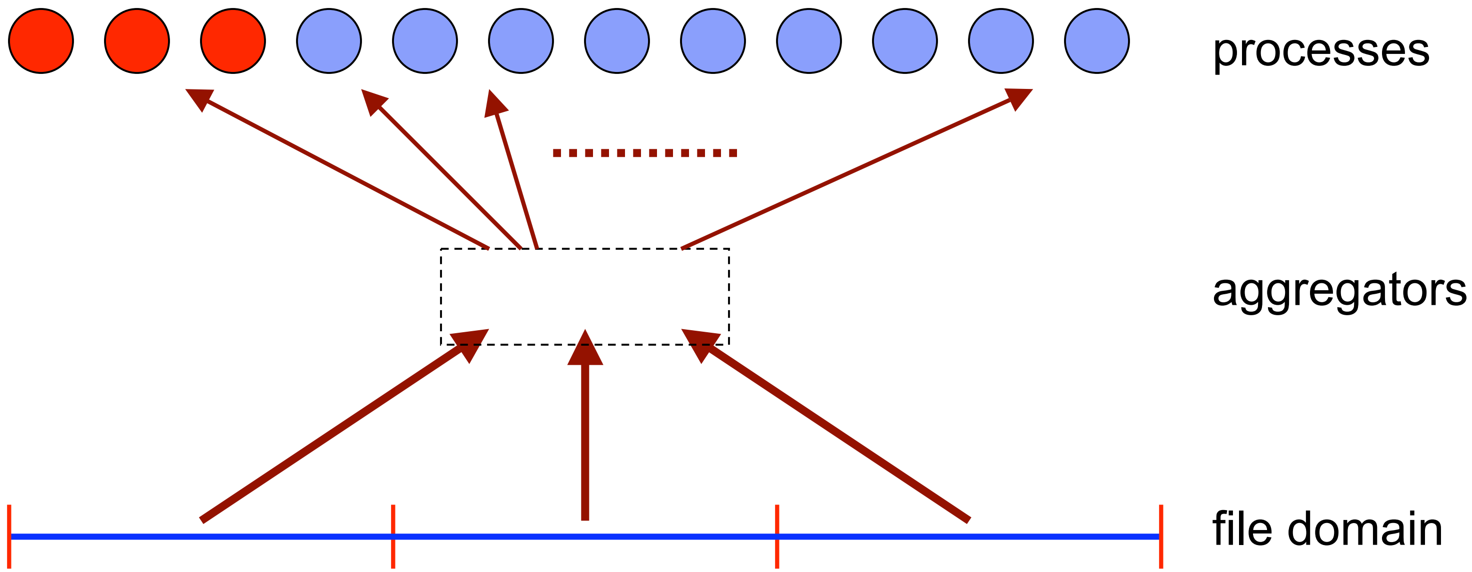


- Reason for the 100 MBps read rate is one-day old driver.
- MPI-IO keeps up with the POSIX-IO perf. for separate file writing and reading.

Collective IO

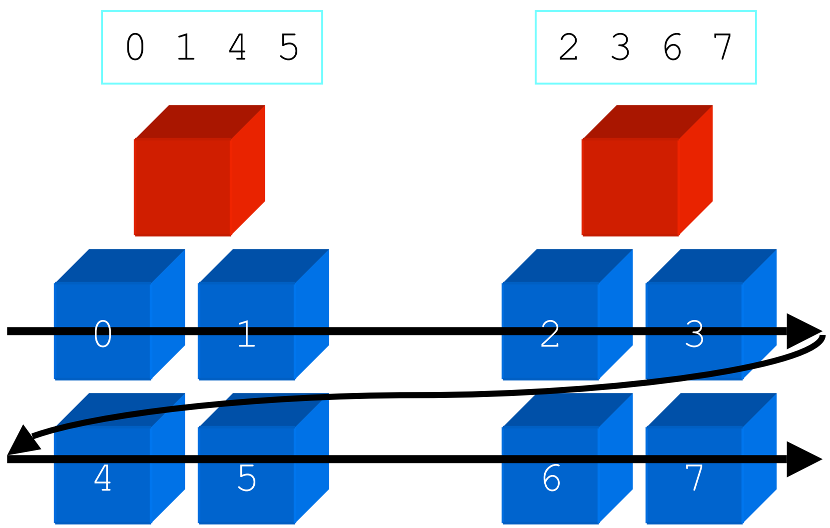
- ROMIO emphasizes on collective IO optimization.
- 2-phase framework
  - Phase 1 aggregates, distributes, and/or redirects IO requests onto a list of IO aggregators by building the communication graph (execution schedule) of the IO requesters and the IO aggregators.
  - Phase 2 carries out the schedule (including data shipping among MPI processes and IO operations from IO aggregators)
  - In this framework, MPI-IO can perform optimizations such as
    - aggregate fine-grain IO requests;
    - balance, distribute IO load among MPI processes.
- For BG/L, we recommend use of collective IO
  - BG/L does not have means to optimize non-collective MPI-IO ops
    - IO node should not be loaded
    - BG/L MPI does not have one-sided comm. Mechanism
    - Look-aside will hurt MPI performance.

Depiction of ROMIO collective read



BG/L specific collective I/O optimizations…

- ROMIO only performs 2-phase IO for non-contiguous IO requests that are not overlapped across processes.
  - On BG/L, compute nodes in a Pset may not have contiguous rank
    - Contiguous and non-overlapped access pattern from application's view-point may become irregular on IO node.
  - We will apply 2-phase IO for contiguous collective IO



IO pattern from IO nodes

Default MPI ranking

… BG/L specific collective I/O optimizations

- ROMIO specifies IO aggregator via user provided hints containing a list of MPI processor names
  - On BG/L, ask a user for such a list will not work
    - Such a list for 1000 processors will take 72KB
    - It is non-trivial for user to generate such a list that is aware of BG/L Pset structures
  - We will provide a hint (bgl_cb_nodes) specifying #IO aggregators in each Pset.

GPFS specific developments

- GPFS file access mode:
  - Default mode: distributed GPFS block level locks are used to provide file consistency.
  - Data shipping mode: accessing a file block has to go through a pre-specified GPFS client node
    - User can distribute file across a set of GPFS client nodes following a cyclic pattern.
    - Need to ship Gpfs_fcntl() from CN to ION.

GPFS specific developments

- ROMIO assumes a regular file domain partition based on a run-time summary of the collective IO operation.
  - Because GPFS' file locking and file distribution are based on a fixed block-size, ROMIO's default file partition may introduce false sharing.
  - GPFS specific or more flexible file domain partitions is considered and corresponding hints shall be provided.
- GPFS only has atomic access mode
  - MPI-IO needs to support relatively efficient atomic access mode.
  - Due to limited power on IO node, such effort is considered in the framework of MPI collective IO.
- Collaborating with Northwestern on these optimizations.

Summary – BG/L MPI-IO is under development

- BG/L MPI-IO is started as a port of Argonne National Lab's MPI-IO implementation, ROMIO.
- Most BG/L MPI-IO operations are functional.
- From preliminary experiments, BG/L MPI-IO seems not introducing much overhead when comparing to POSIX IO.
- We are concentrating on optimizing BG/L MPI-IO for GPFS.
- Collective IO will be the most efficient way to use BG/L MPI-IO.
- Collaboration with ANL ROMIO team and Prof. Alok Choudhary's team at Northwestern U.

Team

- IBM BG/L I/O team: Chris, Parker, Engelsiepen, Volobuev
- IBM BG/L MPI team: Almasi
- Argonne Nation Lab ROMIO team: Ross, Thakur, Latham
- Northwestern Univ: Choudhary, Coloma, Ching
- IBM contact: Yu (yuh@us.ibm.com)

Experiences on Blue Gene at SDSC

Don Thorp

February 23, 2005

Installation went very smoothly

- **Mon Dec 6**
  - Blue Gene single rack arrived & installation began
  - Service node (p275) running Suse 8
  - Front-end nodes (4 B80s) running Suse 9
  - SDSC supplied Cisco switch for Service Net
- **Mon Dec 13**
  - First user got on machine & started acceptance test
  - Machine had to run for three days straight
- **Fri Dec 17**
  - Second user got on machine & performed more tests
  - Machine was accepted

No Problems Next Three Weeks

- **Numerous mpirun bugs & features were found**
  - Classical benchmarks
  - HPC application
- **Security team ran audit**
  - Shared report with IBM

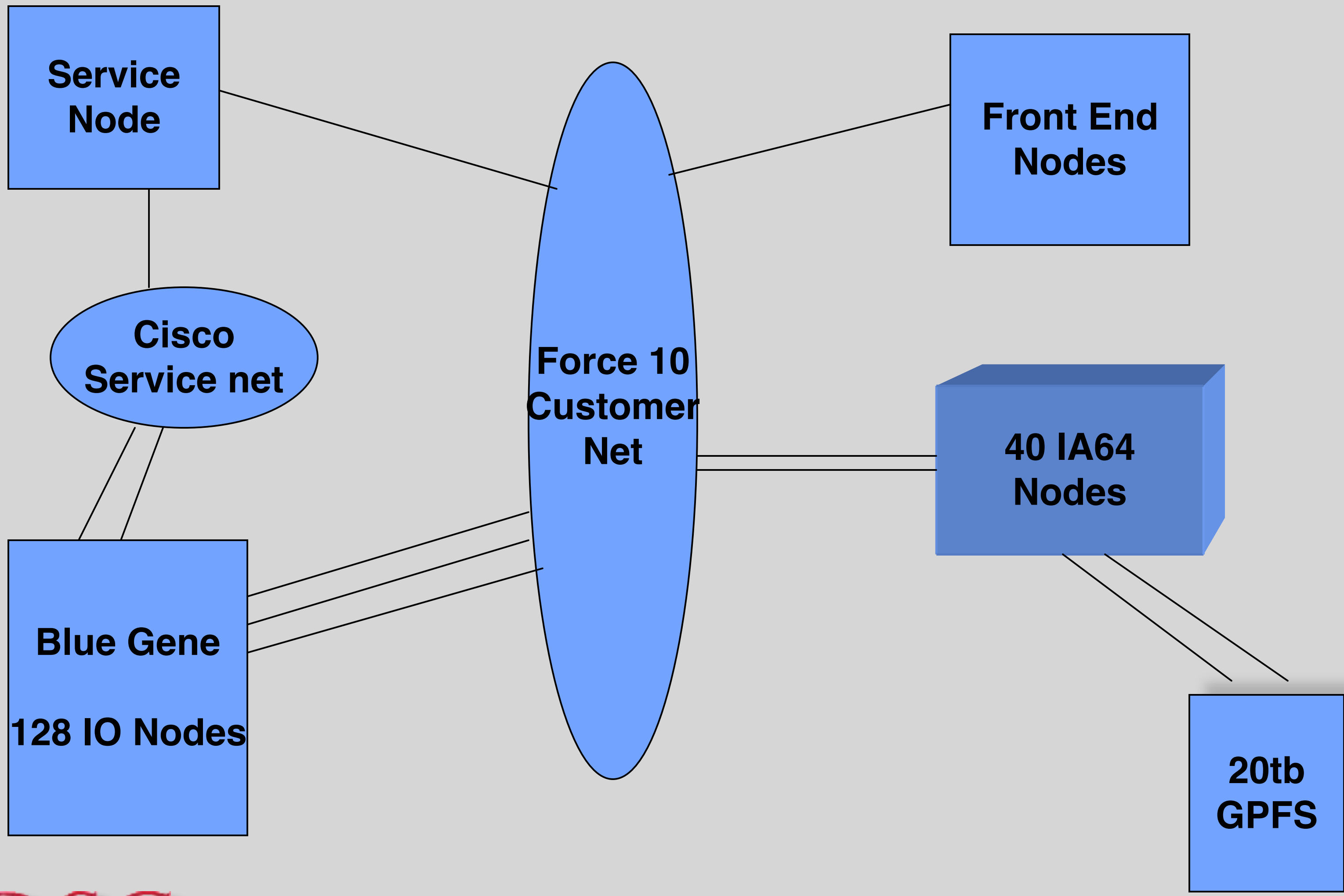
First Troubleshooting Episode during the Second Month

- **On Tue Jan 18 an unscheduled power outage occurred**
- **Two days later machine was brought back up**
  - Software was upgraded from Driver 480 to 521
  - Upper half complained of midplane error
  - Lower half ran until "transaction log" filled
- **Problems were resolved and appear due to**
  - Shutdown procedures corrupting DB2
  - Changed LAN switch setting after reboot; "portfast" disabled
  - Driver 521 software docs incomplete; 480 in /discovery
  - More bad customer cables
  - Increased transaction log buffer to accommodate data loads

Nearterm and Longterm Projects

- **Filesystem Improvements**
  - Replace existing single NFS server with 40 servers
  - GPFS client residing in IO Nodes
- **Tighten Security**
  - Some action required by IBM
  - Some changes in implementation
- **Large IO Benchmarking**
  - Measure maximum IO throughput
- **LoadLeveler**

Network Configuration



Blue Gene/L System Software Update

Kim Cupps

February 23, 2005

UCRL-PRES-210094

Agenda

- Current Status
- Project Timeline
- System Configuration
- Challenges
- What Lies Ahead
- System Reliability
- Early Successes

Blue Gene/L Status
Only Two More Doublings to Go

- We have 16K nodes up and running
- We have another 16K nodes on the floor cabled up, going through individual rack bringup and shakeout process
- Lustre filesystem is being used
  - NFS “pseudo parallel” filesystem will eventually disappear
- We are going to double what we have now **twice** before we finish

Blue Gene/L Integration Timeline

Q1 05 · Q2 05 · Q3 05 · Q4 05 · Q1 06 · . . . Q3 06

09/14 Beta16 70.72 TFLOP Linpack

11/08 1st Four Racks Delivered

12/22 16 Rack System Accepted

1/31 Next 8 Racks Delivered

3/1 – 3/31 System Unavailable Cabling up 32 racks

3/31 32 Rack System Acceptance

6/16 Next 32 Rack Delivery

8/1 – 9/30 System Unavailable Cabling up 64 racks

9/30 64 Rack System Acceptance

10/05 Early Users

Classified BGL BGL LA

- ✓ 12/18 70.58 TFLOP Linpack
- ✓ 1/8 Users on 8K
- ✓ 1/25 User 16K runs
- ✓ 2/4 SLURM and Lustre to users

A Day in the Life of a BGL User

- Familiar environment
  - Login, compile and run on front end nodes
  - xlc, xlC, xlf compilers, debug using TotalView
- Collegial scheduling – everybody talks to each other via instant messaging mechanism
- We plan to run BGL with fixed size partitions that get changed twice a week
  - 16K partition is currently the maximum partition size
  - Currently we run in the (1) 16K job mode 2 days per week for scaling and software debug runs
  - 5 days a week we run 8K, 4K, 2K, and (4) 512 node partitions
- Currently all **Compute Nodes** and I/O nodes must be re-booted between jobs
  - This takes about 8 minutes for 16K partition, including mounting Lustre

BG/L System Configuration Overview

- Blue Gene/L Core = Compute Nodes + IO Nodes
- Compute Nodes
  - 64 racks
  - 1K compute nodes per rack
- IO Nodes
  - 16 IO nodes per rack
  - 1024 IO nodes
- Compute and IO Node specs
  - dual-processor PPC 440 @ 700Mhz
  - 512 MB DDR memory
  - 4 MB L3 cache
- Compute Node : IO node ratio
  - 64:1 ratio is higher than most other BGL systems

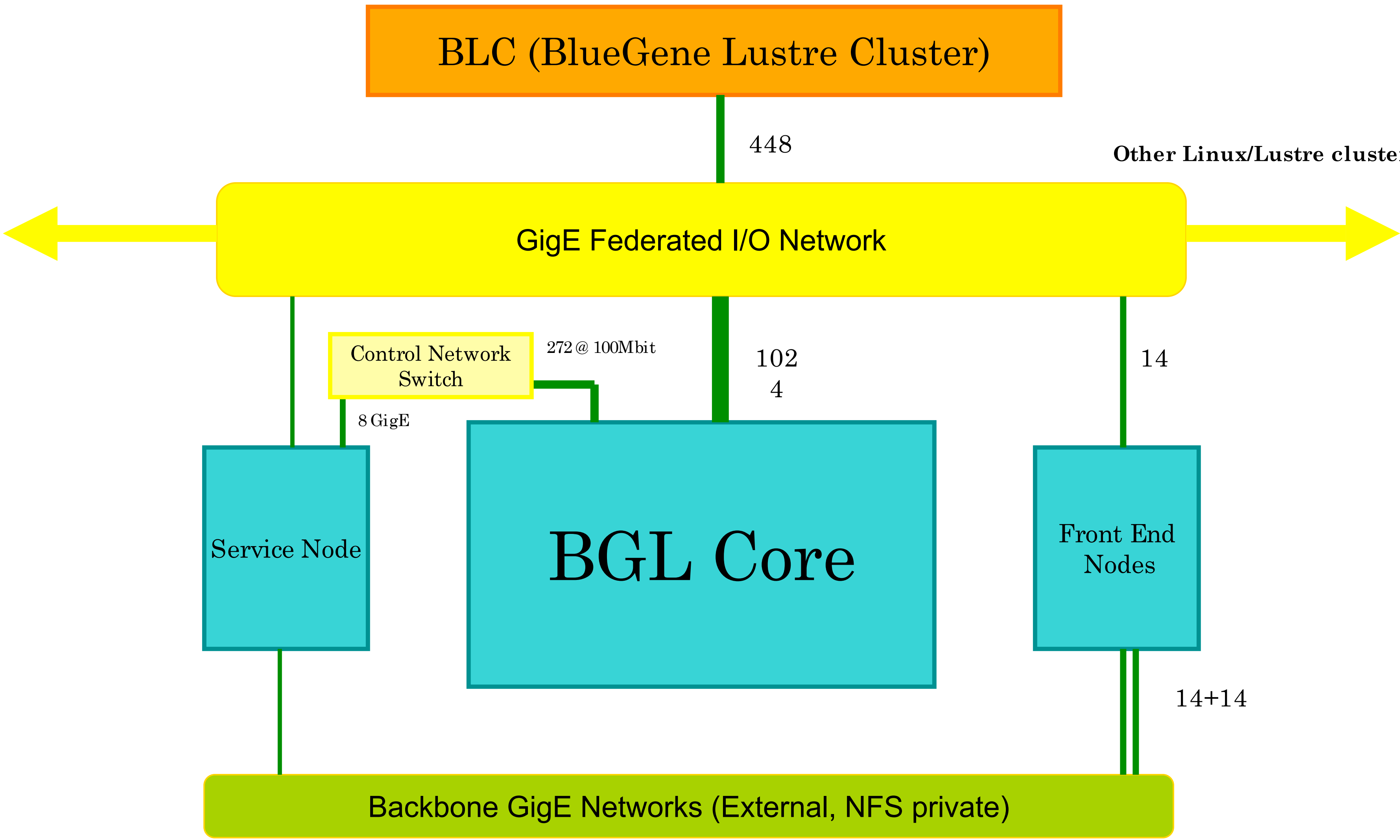
System Configuration Overview (cont'd)

- Service node: IBM p670
  - 16 Power4+ processors, 64 GB memory
  - Runs SuSE SLES 8

- Front end nodes: BladeCenter JS20
  - Fourteen dual-processor PPC 970 blades @ 1.6GHz
  - Runs SuSE SLES 9

- BLC = Blue Gene Lustre Cluster
  - 224 dual-processor Intel EM64T "Nocona" OST nodes at @ 2.8 Ghz
  - ~900 TB Lustre filesystem
  - Target delivered I/O bandwidth of 32GB/s to user apps (32MB/s/ION)

System/Network Layout



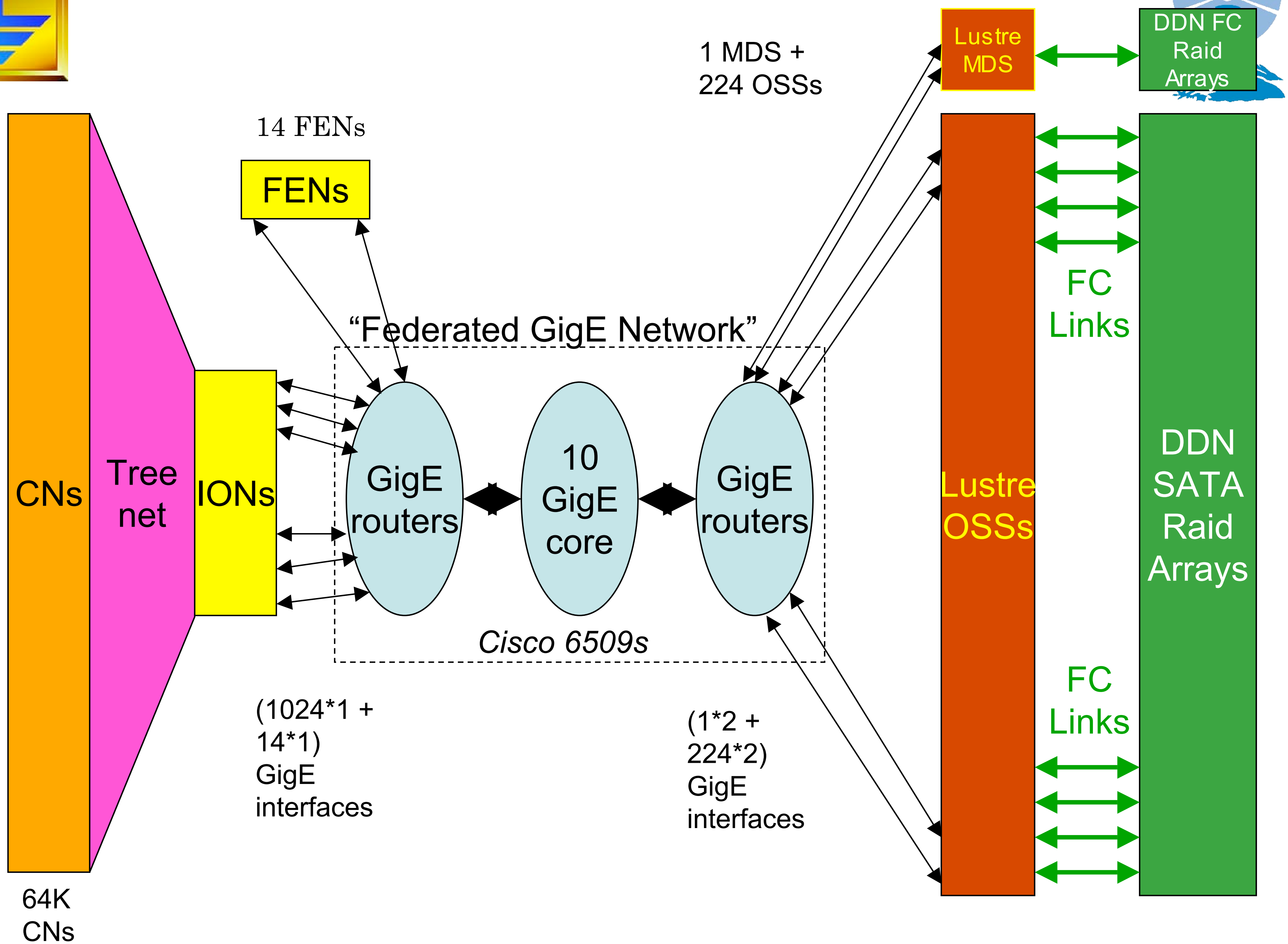
Challenges

- Port Lustre to I/O Nodes and Front End Nodes (FENs)
- Port SLURM to BG/L
- Port TotalView to BG/L
- Operational Challenges

BG/L Lustre Integration Challenges

- Server side infrastructure (224 OST nodes, 900 TB back end storage) is operational but only 1/3 of it has been used enough to shake out HW issues.
- Port of Lustre to JS20 front end nodes (PPC64, SLES9) is complete, stress testing in progress, performance looks good with SLES9 2.6 kernel.
- Port of Lustre to BGL IO nodes (PPC32, 2.4.19 kernel) is functionally complete, but many issues still need to be addressed:
  - Several issues have been debugged both in IBM and CFS software
  - Performance is improving on a weekly basis
  - Time to mount Lustre after partition reboots was slow but has been improved (< 1 minute for 16K nodes)
  - Failure of a single Lustre client on IO node requires reboot of entire partition. BGL has no provision to reboot a single IO node.
  - Scaling and stability issues are likely still ahead…

Full BG/L Lustre “Big Picture”



Resource Management Challenges

- Simple Linux Utility for Resource Management (SLURM) can define, create and destroy partitions (via *smap, slurmctld* on service node), and queue/run user jobs (via *slurmd, srun* on front end nodes)
- Outstanding issues in IBM software stack:
  - Partition must be rebooted any time a different user’s job is launched.– so running static partition sizes doesn’t prevent reboots.
  - Partition reboot required when changing from Virtual Node Mode to Co-Processor mode even under the same user
- LCRM ported but not tested

Operational Challenges

- Five different Operating Systems
- First experience with DB2 on Linux
- First experience with SuSE Linux
  - Need to port existing system management tools to SLES8/9
  - Lustre port to SLES9 FENs requires kernel modifications.
  - General lack of responsiveness by SuSE support
  - Figuring out a patch strategy: need to address security vulnerabilities but avoid breaking BGL software stack.
- Security issues with BGL control system software stack
  - IBM is performing an internal audit of this software.
- Support
  - IBM's BGL support infrastructure still being defined
  - Two hour IBM response time between 6 AM and 3 PM, no off-hours IBM support
  - Single point of contact via telephone
  - Problem tracking and reporting system not fully functional

Code Development and Scaling Challenges

- TotalView port is progressing well, issues are being reported to IBM
  - TV/mpirun doesn’t work in virtual node mode
  - TV/mpirun doesn’t work when number of tasks is less than full partition size
- MPI functionality and performance issues being tracked and reported to IBM
- Compiler issues are being tracked and reported to IBM
- Code scaling effort progressing well

System Reliability

- Hardware infant mortality rate is slightly less than one might expect
  - 6 compute node failures, 2 IO node failures since IBM install team left on 12/20.
- System Software stack is still under development
  - New drivers in development at IBM every 2 weeks
  - Major functionality changes between driver versions (including API changes)
  - Control system software is least mature

BGL Accomplishments

- 16K Linpack result achieved five weeks after delivery of the first rack.
- Functional acceptance of 16K system achieved six weeks after delivery of the first rack.
- RAS Database is easy to use and fairly robust
- Users running REAL SCIENCE on the system shortly after acceptance test completion

What's Ahead

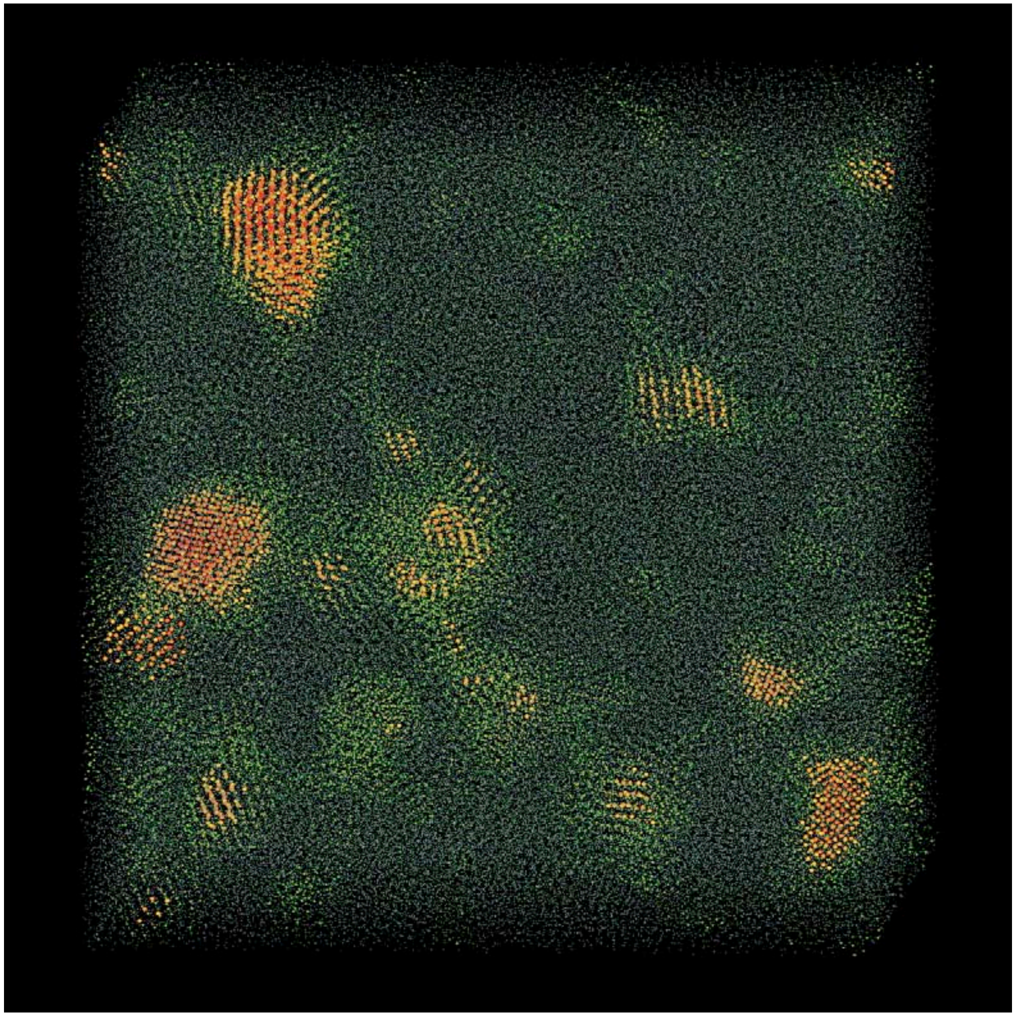
- Integration, integration, integration
- Test, test, test
- System will definitely run Pu aging codes in classified environment – Q3 2006?
- We want a 1 rack unclassified system but no funding has been identified
- Pursuing approval for swinging the machine from classified to unclassified
  - If approved we plan to swing the compute and IO nodes about 3-4 times per year

Ground Breaking Science

"You are looking at 256,000 tantalum atoms, with size and color determined by a local order parameter which has picked out several regions which are solidifying. The simulations starts with cell of molten Ta at 5000K and is then compressed (not frozen) into being a solid at about 2 Mbar of pressure. Most of the atoms are grey-green and very small, varying smoothly to red and large for highly ordered atoms. Displayed in this way, you can readily see the regions that are ordered. These nucleation sites have formed spontaneously out of the melt and will continue to grow, eventually forming a solid chunk of Ta with various grain boundaries. We've simulated the whole process.

"What Blue Gene has enabled us to do is perform this simulation on a large enough sample to see multiple such nucleation sites, so that we can (a) better understand how they form and (b) study how they interact as they grow. Previous simulations were limited to several thousand atoms, which corresponds to a cell size roughly the size of one of those nucleation sites. Needless to say, this has severely limited our ability to study the larger problem. It should be pointed out that many people have performed simulations using computationally cheap pair-wise potentials involving millions or even billions of atoms - but this is the first simulation on this scale using the very accurate but expensive MGPT potential, which is a quantum based potential."

Frederick H. Streitz
LLNL Rapid Resolidification Project



Summary

- We have done a lot of work and it's looking pretty good
- We aren't quite half way there
  - We have a long way to go
- This is the hardest thing we've ever attempted and we are right in the middle of it!

Early Experiences with BG/L

Susan Coghlan
<smc@mcs.anl.gov>
Argonne National Laboratory

MCS BG/L Specs

- 1 Rack (1024 nodes)
- 32 I/O nodes (1/32 IO/Compute ratio)
- 4 Frontends (JS20 blades – PPC970 2.1GHz dual-cpu 4GB RAM) [SLES9]
- 1 Service Node (4-way 1.7 Ghz PPC (2 CPU cores), 16GB RAM) [SLES8]
- 20 Storage servers (4 homedir, 16 PVFS) ~14TB [SLES9]

Time Line

- Install started 1/20/2005
- Linpack ran successfully 1/21/2005
- Acceptance delayed
  - Waiting for storage nodes and frontends
  - Problems with one of our applications
- Machine accepted 1/31/2005
- Users on running apps 2/2/2005

Installation Observations

- BG/L Installation is clearly a WIP
  - We benefited from LLNL and SDSC installations
- Having a process is useful, but
- Generally went well
- The hardware is not the problem

Installation Issues

- Hardware Problems (Minimal)
  - 2 Compute nodes (+1 last week)
  - 4 I/O nodes
- Configuration Quirks, things like:
  - Must define 64 I/P for I/O nodes even though we only have 32
  - Delete/reload block definitions, then double allocate required after boot
- Software Problems
  - everyone needs a FLASH in their acceptance test suite

The Mission

- System Software Development
- Scientific Application Porting
- Performance Testing and Benchmarking
- Community Resource

- Our operation at ANL must be extremely flexible!

Making it Usable

- System Software Modifications
  - Filesystem
  - Resource Manager
  - Partition Management – ANL
  - I/O Node Environment
- Rational User Environment
- System Management Needs

Filesystems

- NFS – make sure your rc.d script(s) retry enough times
- Other Filesystems [LLNL (Lustre), SDSC (GPFS) and ANL (PVFS2)]
- PVFS running as a production filesystem on Jazz for the past 2 years
- Mounted across full BGL rack
- Performance tuning in progress

Resource Management

- LLNL(SLURM), SDSC(LoadLeveler), ANL (no-name)
- Developed at MCS, in use on Chiba
- Opensource components based on the SSS component interfaces
- Lightweight implementations written in python
- BGL Components:
  - Process Manager (process startup and control)
  - Queue Manager
  - Scheduler (currently simple FIFO)
  - Allocation Manager (integration not completed)
- In Alpha Testing

ANL Ramdisk and I/O kernel

- ANL Linux I/O node toolkit available for distribution
- Linux kernel, config, compile & ramdisk tools, etc.
- Open source distribution of the I/O node ramdisk and kernel
- We are currently using it to extend the capabilities of the I/O node, and to build performance tools (TAU) and kernel modules (PVFS)

Building the User Env

- As installed, not robust (csh broken)
- Installed Softenv (developed at MCS)
- Moved all the BGL-isms under Softenv
- Integrated into MCS account management
- Prepared partitions
- Created mail lists – discuss and notify
- Built a status page

Tools for the Users

- Bgl-list
  - List jobs: running, errored, completed
  - List blocks: allocated, all
  - Long listing: all data found (* location)
  - By ID, * by user, * by partition
- * Tool for processing logs, RAS events
- * Tool for cleaning up hung jobs
- * Tool for managing mapping

Current State

- Operating mode:
  - 32 node 'developer' partitions
  - Co-processor and Virtual mode versions
  - Small set with personal ramdisks and kernels
  - Evenings/weekends 512 and 1024 node runs
- 26 active users
- ~3000 jobs run (80% in developer)

A Few of the Projects

- nQMC
- NeoCortex
- Nanocatalysis
- QCD
- POP
- MPIBlast
- TAU/PDT
- PETSc
- MPE/Jumpshot

Early User Experiences (what they liked)

- Code compiled and ran first time
- Fast communications
- Nice scaling
- Ability to map processes to underlying topology is cool (if I could get it to work)
- It's certainly a challenge

Early User Experiences (what they didn't like)

- Lack of useful information
- Lack of documentation
- Compiler is buggy, options don't work as expected and diagnostics are poor
- Lack of a simple mpirun that just works. Options don't work as expected.
- Debugging is Hard!

Support So Far

- Not thrilled with the support interface
- Not thrilled with the response times
- Quick solutions once we made contact
- We haven't pushed hard, yet
- but
- our bug list is growing rapidly
- some are most likely fixed in newer versions

ANL BG/L Community Resources

- ANL BG/L Wiki (available now)
  - http://www.bgl.mcs.anl.gov/wiki
- MCS BG/L Web site (available now)
  - http://www.bgl.mcs.anl.gov
- Problem tracking system (available soon)
  - http://www.bgl.mcs.anl.gov/support

Prioritized List – Users View

- Debugging tools
- Software upgrades – compiler, mpi
- Fully functional compiler with good diagnostics and correct assembly code
- Fully functional mpirun
- Documentation, all sorts, more of.
- Documentation that matches reality
- PAPI, etc.

Users list, continued

- Mapping documentation
- Usable error information
- Correct date/time and timing mechanisms that go beyond 7hr18m (MPI_GET_TIME, Fort DATE_AND_TIME, cpu time)
- Dynamic libraries
- MIMD support

Prioritized List – SysAdmin View

- Quick access to error translations to understand failure modes
- Simplified startup process
- No VNC requirement (VNC – big mistake)
- DB2 – rational protections, documentation for relations, schema, etc.
- Documented interface RAS/etc [for automated monitoring, i.e. nagios]

SysAdmin list, cont.

- Up-paced software updates (with better revision numbering)
- Better support model
- Rational I/O node environment
- Source for ciod,mpi,mmcs_*

A Tutorial on BG/L Dual FPU Simdization

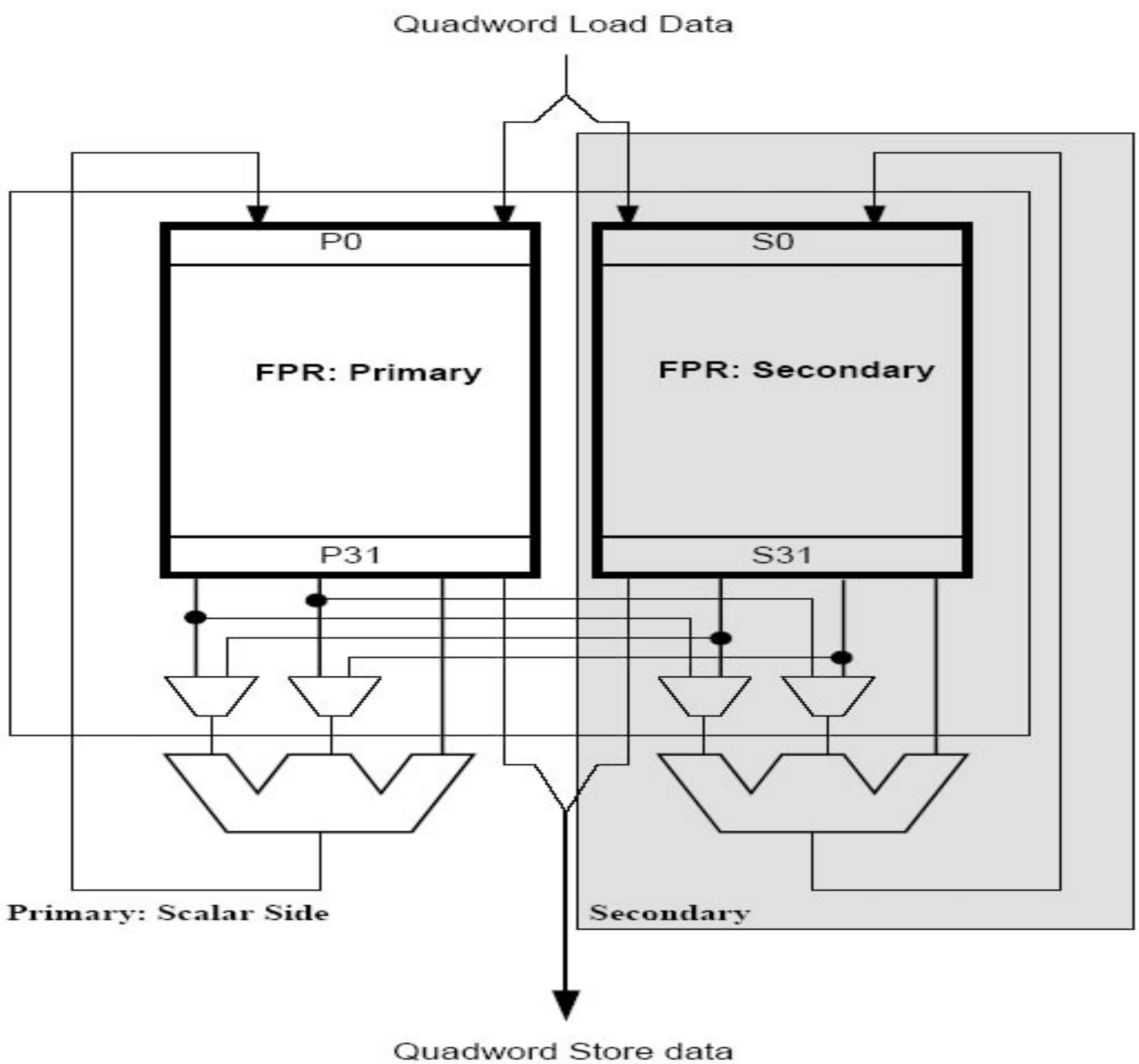
Alexandre Eichenberger, Rohini Nair, and Peng Wu / TPO

Mark Mendell / Tobey

Outline

- Background
- How to use the compiler
- Diagnostic info and tuning
- Alignment handling
- Experimental results

BlueGene/L Dual Floating Point Unit



Architecture Constraints of Dual FPU Unit

- Only stride-one memory accesses use full bandwidth
  - "stride-one" means "stored consecutively in memory"
  - lower bandwidth for non-stride-one accesses (non major, a[2i+1], indirect access

- Access efficiently only 16-byte aligned data
  - a[i] = b[i] +c [i]   vs.   a[i] = b[i+1] + c[i]

- Misaligned data can be loaded using cross-instructions
  - data realignment pattern is encoded in the instructions,
  - makes handling of runtime alignment difficult

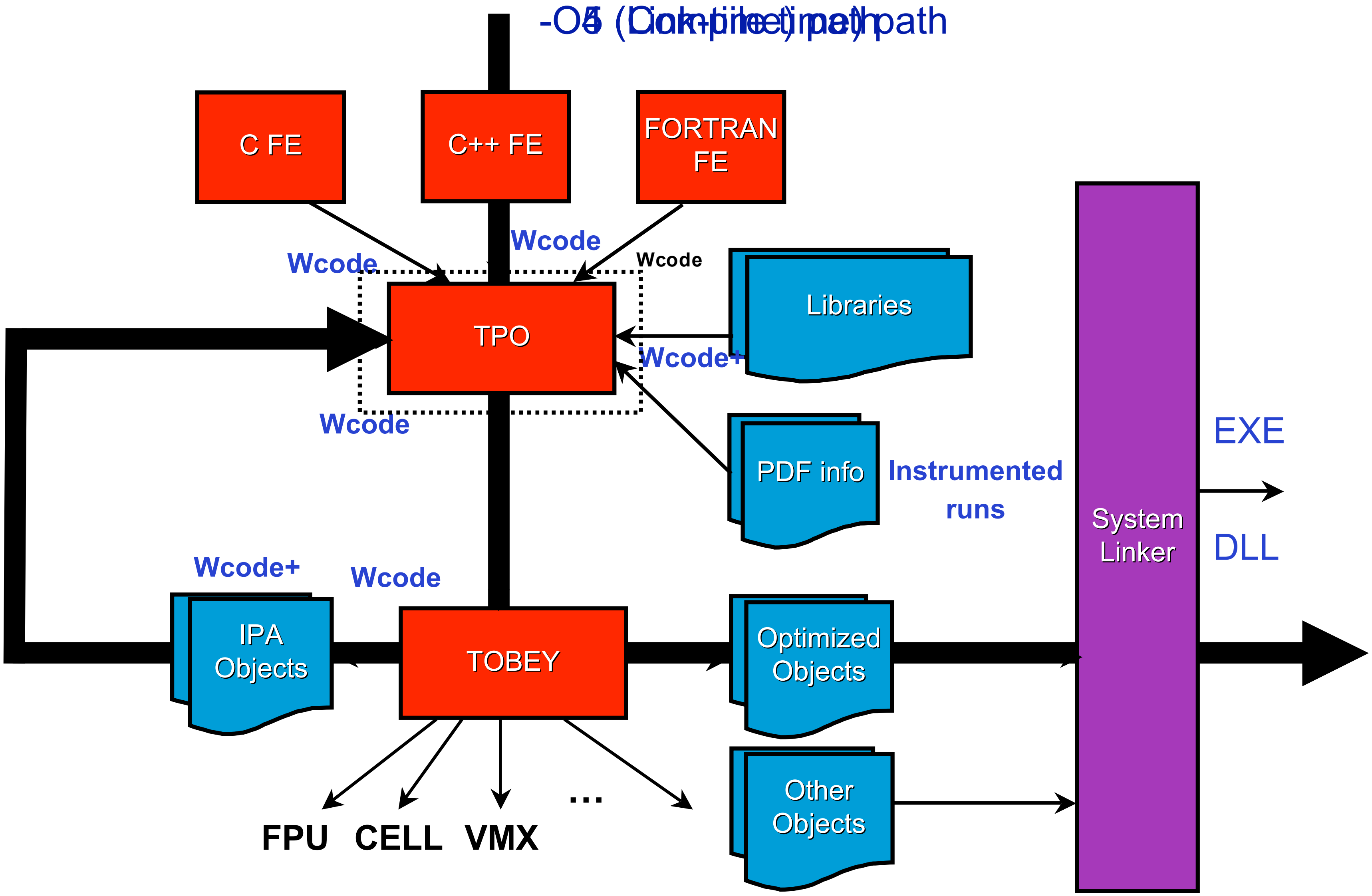
- Non-uniform instruction set for dual unit
  - double precision floating point only

- Simdization → SIMD vectorization

Outline

- Background
- How to use the compiler
- Diagnostic info and tuning
- Alignment handling
- Experimental results

The XL Compiler Architecture



Where does Simdization Occur?

- Some occurs in TPO (high-level inter-procedural optimizer)
  - computations that stream over double floats
  - TPO does most loop level/inlining/cloning optimizations

- Some occurs in Tobey (low-level backend optimizer)
  - complex arithmetic on double floats is an ideal target
  - other non-regular double floats are also packed
  - Tobey does most code motion/scheduling/machine specific optimizations

This talk focus mainly on TPO level simdizatio

3-Step Program to Enable Simdization

- Compile for the right machine
  - -qarch=440d –qtune=440 (in this order)

- Turn on the right optimizations
  - -O5 (link-time, whole-program analysis & simdization)
  - -O4 (compile time, limited scope analysis & simdization)
  - -O3 –qhot=simd (compile time, less optimization & simdization)

- Tune your programs
  - use TPO compiler feedback (-qxflag=diagnostic) to guide you
  - help the compiler with extra info (directive/pragmas)
  - modify algorithms (hint: more stride-one memory accesses)

2-Step Program to Disable Simdization

- Compile for the wrong machine
  - to completely disable simdization: -qarch=440 –qtune=440

- Turn off the right optimizations
  - compile for –qarch=440d –qtune=440

  - disable TPO simdization (keep Tobey simdization)
    - for a loop: #pragma nosimd | !IBM* NOSIMD
    - completely: -qhot=nosimd

  - disable Tobey simdization (keep TPO simdization)
    - not supported, may not work, try at your own risks
    - completely: -qxflag=nhummer:ncmplx

green is for C | red is for for

5 to 7 Steps to Help Us

- Found a correctness bug?
  - play with options to see at which level it fails
  - isolate the error (code as small as possible)
  - simdize only the loop that fails
  - give us all the info (all sources, header, make files, compiler options)
  - report the problem

- Found a performance bug?
  - test the correctness of your code (verify results if possible)
  - try to estimate a good lower bound (number of mem/fma/…)
  - apply above 5 steps

Outline

- Background
- How to use the compiler
- Diagnostic info and tuning
- Alignment handling
- Experimental results

Examples of TPO Simdization Success Diagnostic

Examine loop <1> on line 12
(simdizable) []

Examine loop <2> on line 20
(simdizable) [misalign(compile time) shift(3 compile-time)]

Examine loop <3> on line 26
(simdizable) [misalign(runtime)][versioned(relative-align)]

TPO Diagnostic Information on Success

- Simdizable loops
  - diagnostic reports "`(simdizable) [features] [version]`"

- [feature] further characterizes simdizable loops
  - "`misalign(compile time store)`": simdizable loop with misaligned access
  - "`shift(4 compile time)`": simdizable loop with 4 stream shift inserted
  - "`priv`": simdizable loop has private variable
  - "`reduct`": simdizable loop has a reduction construct

- [version] further characterizes if/why versioned loops where created
  - "`relative align`": versioned for relative alignment
  - "`trip count`": versioned for short runtime trip count

-qxflag=diagnostic report on TPO Simdization o

Examples of TPO Simdization Failure Diagnostic

Examine loop <id=1> on line 1647
    not single block loop
(non_simdizable)

Examine loop <id=1> on line 2373
    dependence at level 0 from (0 73 100)
(non_simdizable)

Examine loop <id=2> on line 2356
    dependence due to aliasing
(non_simdizable)

Examine loop <1> on line 4
    no intrinsic mapping for <ADD int>: a[]0[$.CIV0] + b[]0[$.CIV0]
(non_simdizable)

TPO Diagnostic Information on Failure

- **Alignment:**
  - "`misalign(….)`": simdizable loop with misaligned accesses
    - "`non-natural`": non naturally aligned accesses
    - "`runtime`": runtime alignment

⇒Action:

- align data for the compiler: double a[256] **__attribute__((aligned(16));**
  - all dynamically allocated memory (malloc,alloca) are 16-byte aligned
  - all global objects are 16-byte aligned
  - inside struct / common block, you are on your own
- tell the compiler it's aligned: __alignx(16, p); | call alignx(16,a[5]);
  - like a function call, no code is issued
  - can be placed anywhere in the code, preferably close to the loop
- tell compiler that all references are naturally aligned
  - -qxflag=simd_nonnat_aligned
- use array references instead of pointers when possible

green is for C | red is for for

TPO Diagnostic Information on Failure (Cont')

- Structure of the loop
  - `"irregular loop structure (while-loop)"` (handle only for/do loops)
  - `"contains control flow"`: (no if/then/else allowed)
  - `"contains function call"`: (no function calls)
  - `"trip count too small"`: (short loops not profitable)

⇒Action:

- convert while loops into do loops when possible
- limited if conversion support
  - handle best if-then-else with same array defined on both sides
  - can try data select
- inline function calls
  - automatically (-O5 more aggressive, use inline pragma/directives)
  - manually

TPO Diagnostic Information on Failure (Cont')

- Dependence
  - `"dependence due to aliasing"`

⇒Action:

- help the compiler with aliasing info
  - use -O5 (does interprocedural analysis)
  - tell the compiler when its disjoint: #pragma disjoint (*a, *b)
  - use fewer pointers when possible

- Scalar references
  - `"non-simdizable reductions"`
  - `"non-simdizable scalar var"`

⇒Action:

- reductions that are used in the loops can not be simdized

TPO Diagnostic Information on Failure (Cont')

❑ Array references

- "`access not stride one`":
- "`mem accesses with unsupported alignment`"
- "`contains runtime shift`"

⇒Action:

- interchange the loops to enhance stride-one, when possible
- sometime TPO may interchange loops for you, in a way that you don't want
  - disable unimodular transformation: -qxflag=nunimod
- runtime alignment not feasible on BG/L
  - compiler version the loop
  - one of the two version may report "`(non-simdizable)`"

TPO Diagnostic Information on Failure (Cont')

- Pointer references
  - "`non normalized pointer accesses`"

⇒Action:

  - simple pointer arithmetic should be well tolerated
  - otherwise, try using arrays

- Native Mapping and native data types
  - "`non supported vector element types`"
  - "`no intrinsic mapping for <op type>`:"

⇒Action:

  - none: BG/L supports only double precision floating point SIMD

Other Tuning

- Loop unrolling can interact with simdization
  - there is some support for simdizing unrolled loop, but its harder
  - try to not manually unroll the loop for better TPO simdization
  - unroll directive: #pragma nounroll | #pragma unroll(2)

- Math libraries:
  - currently, we don't simdize sqrt,…
    - we split the loop, simdize the one without sqrt
    - you can do the same, short loop that compute all the sqrt, store in a temp ar
    - use optimized libraries to compute vectors of sqrt
    - then use it in the old loop, that one will simdize

- Use literal constant loop bounds
  - e.g. #define when possible

- Tell compiler not to simdize a loop if not profitable (e.g., trip count too low)
  - #pragma nosimd (right before the innermost loop)

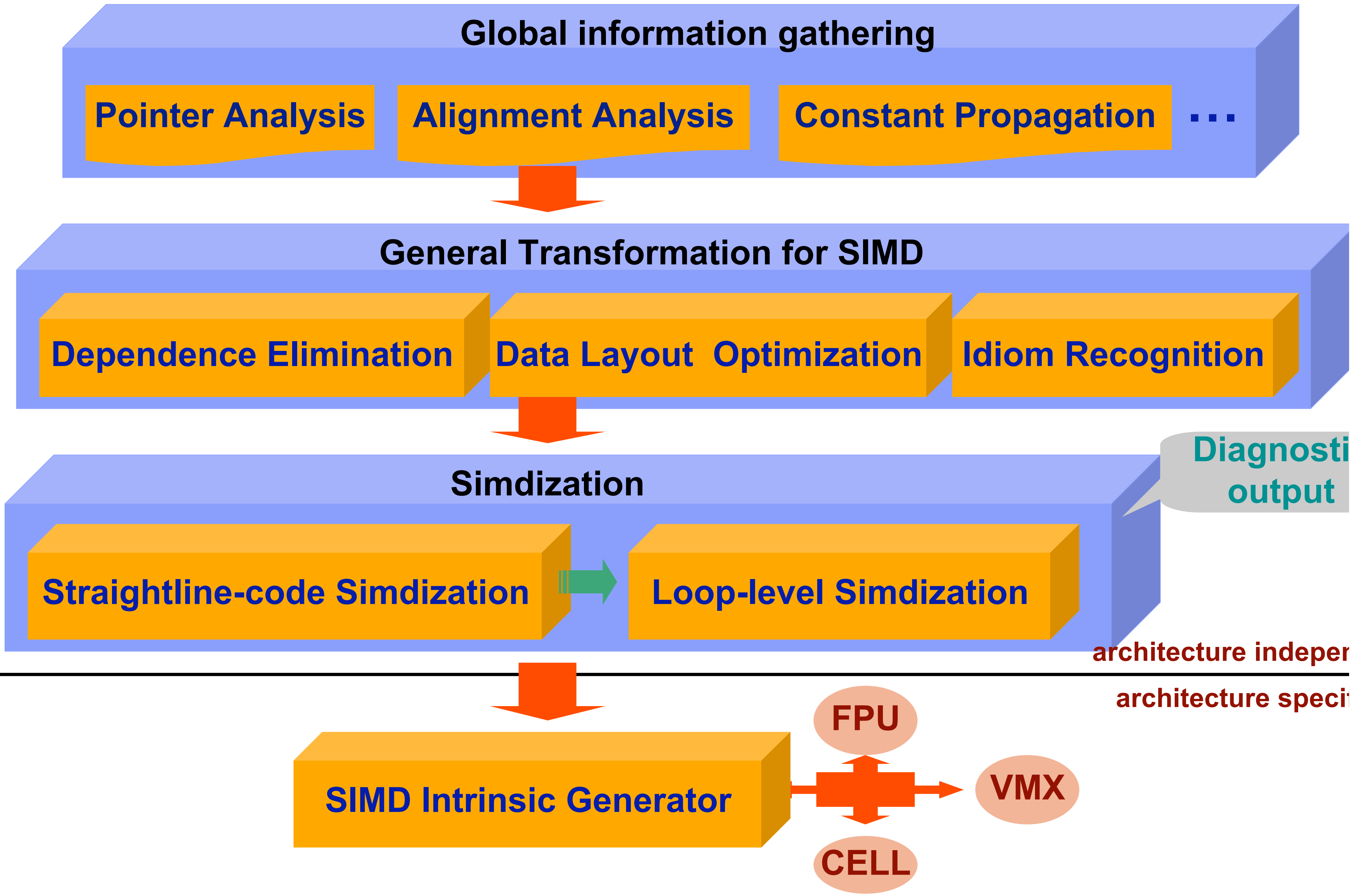
More pragma/directive info

- Some generally available info is here
  - http://publib.boulder.ibm.com/infocenter/comphelp/index.jsp
  - some useful links on this site:
    - Fortran/Language references/Directives
    - Fortran/Language references/Intrinsic procedures/Hardware specific
    - C/Language references/Preprocessor directives/Pragma directives

Outline

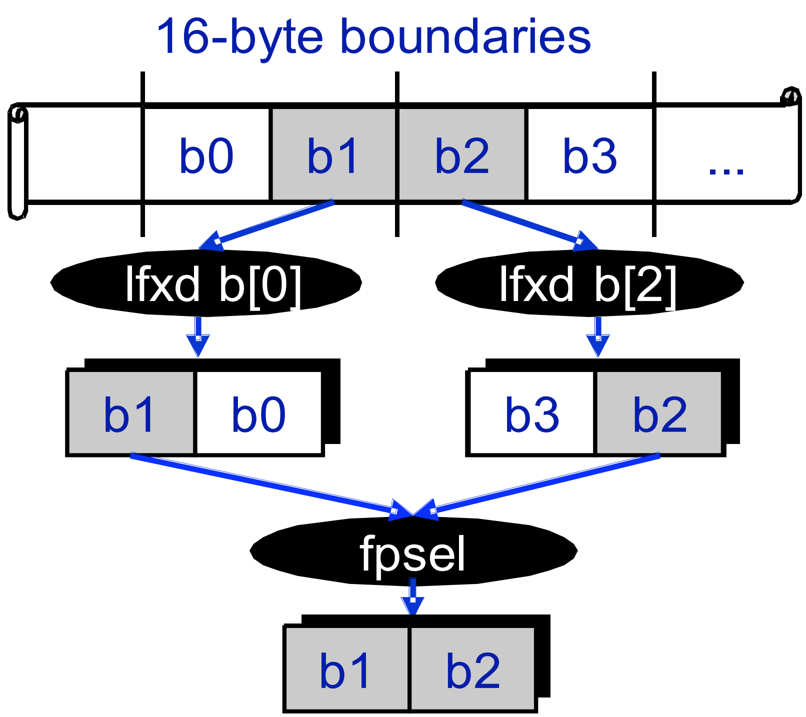
- Background
- How to use the compiler
- Diagnostic info and tuning
- Alignment handling
- Experimental results

A Unified Simdization Framework



How to load from misaligned memory?

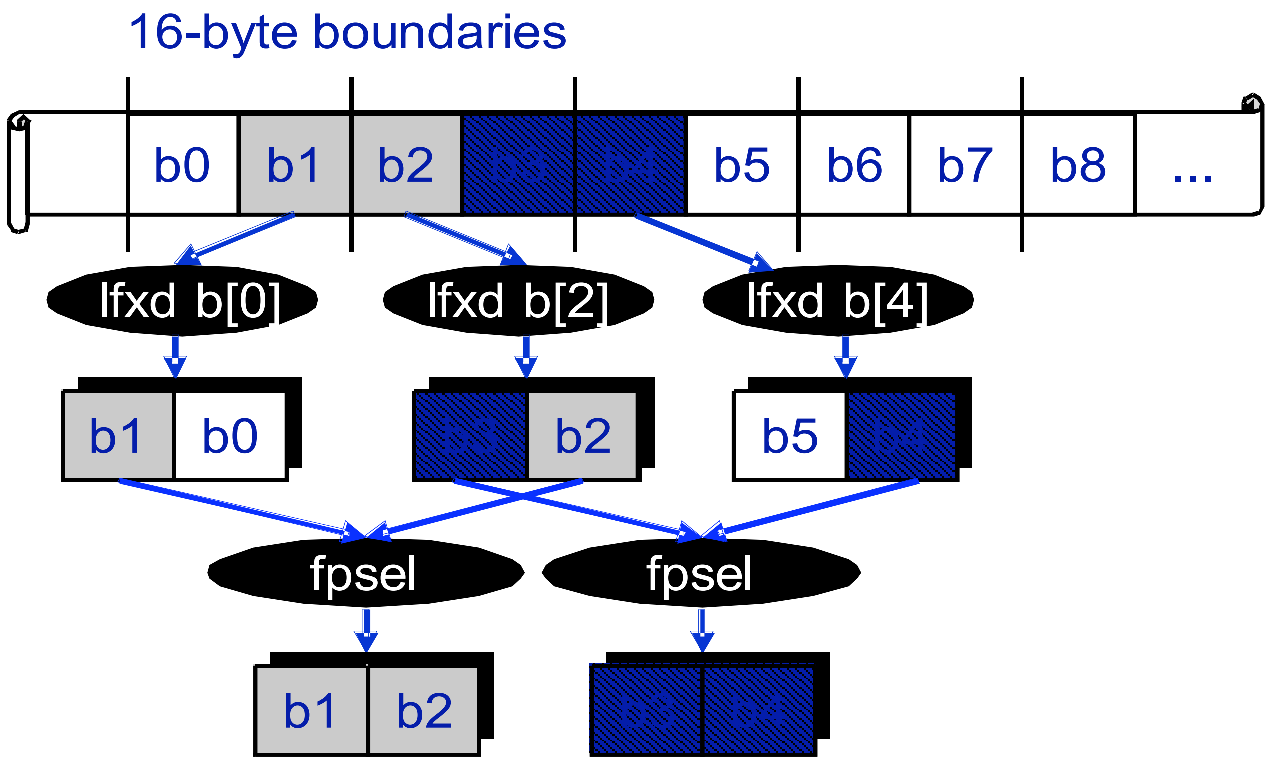
- Load one misaligned quad:



1 misaligned-quad load costs 2 aligned-quad cross-loads + 1 select

How to access misaligned memory (cont')?

- Load multiple consecutive misaligned quad data:
  - reuse quad load-across



1 misaligned-quad load costs on avg. 1 aligned-quad cross-loads + 1 sele

When misalignment handling is needed?

- for (i=0; i<100; i++) a[i] = b[i] + <u>c[i+1]</u> ;
  - aligned: a[i], b[i]
  - misaligned : c[i+1]
  - action: realign c[i+1]

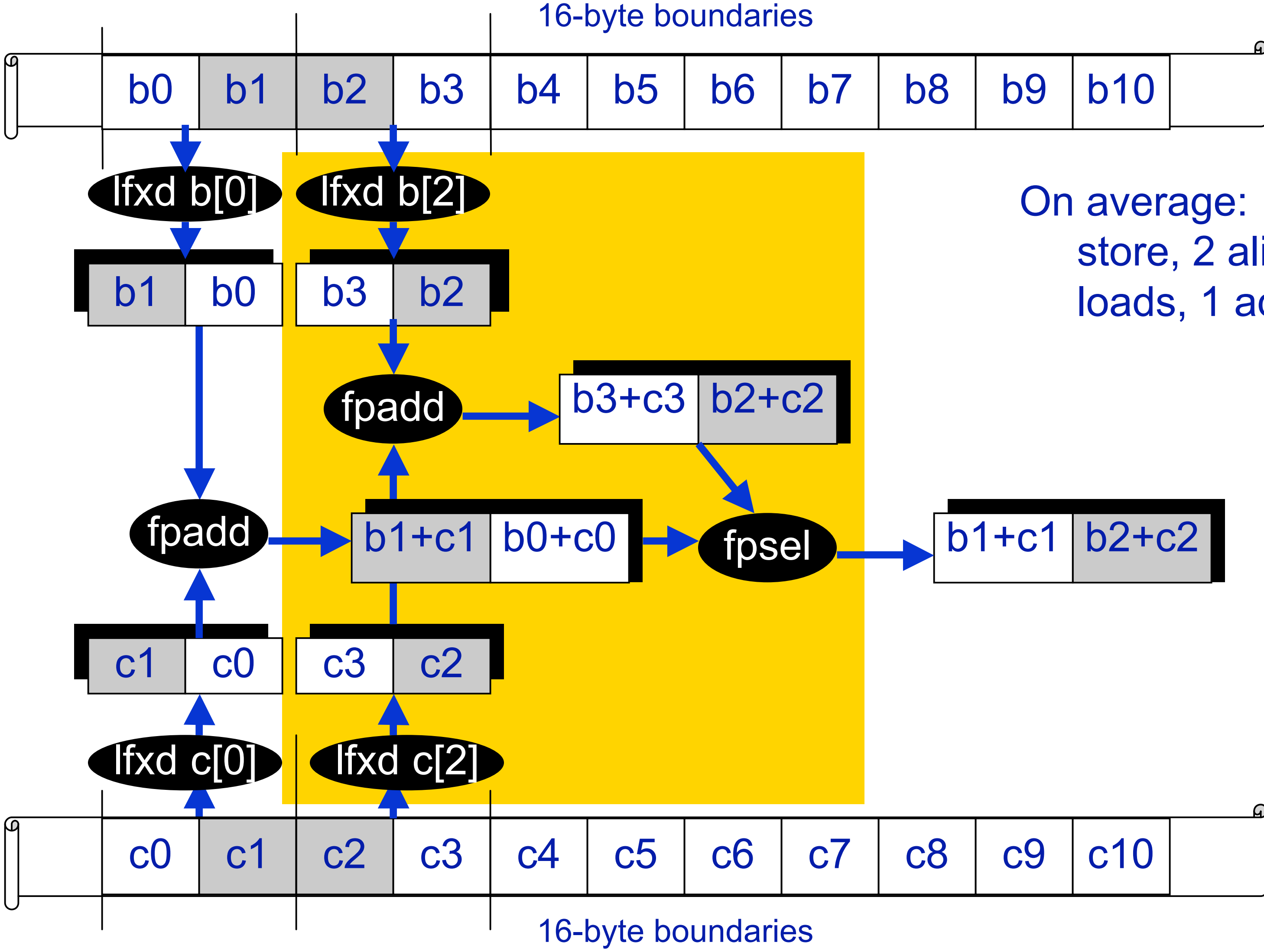
- for (i=0; i<100; i++) <u>a[i+1]</u> = <u>b[i+1]</u> + <u>c[i+1]</u>;
  - misaligned, but relatively aligned: a[i+1], b[i+1], c[i+1]
  - action: peel first iteration

- for (i=0; i<100; i++) <u>a[i+1]</u> = <u>b[i+1]</u> + c[i];
  - misaligned, but relatively aligned: a[i+1], b[i+1]
  - aligned: c[i] is aligned
  - action: peel first iteration, realign c[i]

a[0], b[0], c[0] assumed aligi

Minimizing data reorganization overhead

- for (i=0; i<100; i++) a[i] = b[i+1] + c[i+1] ;

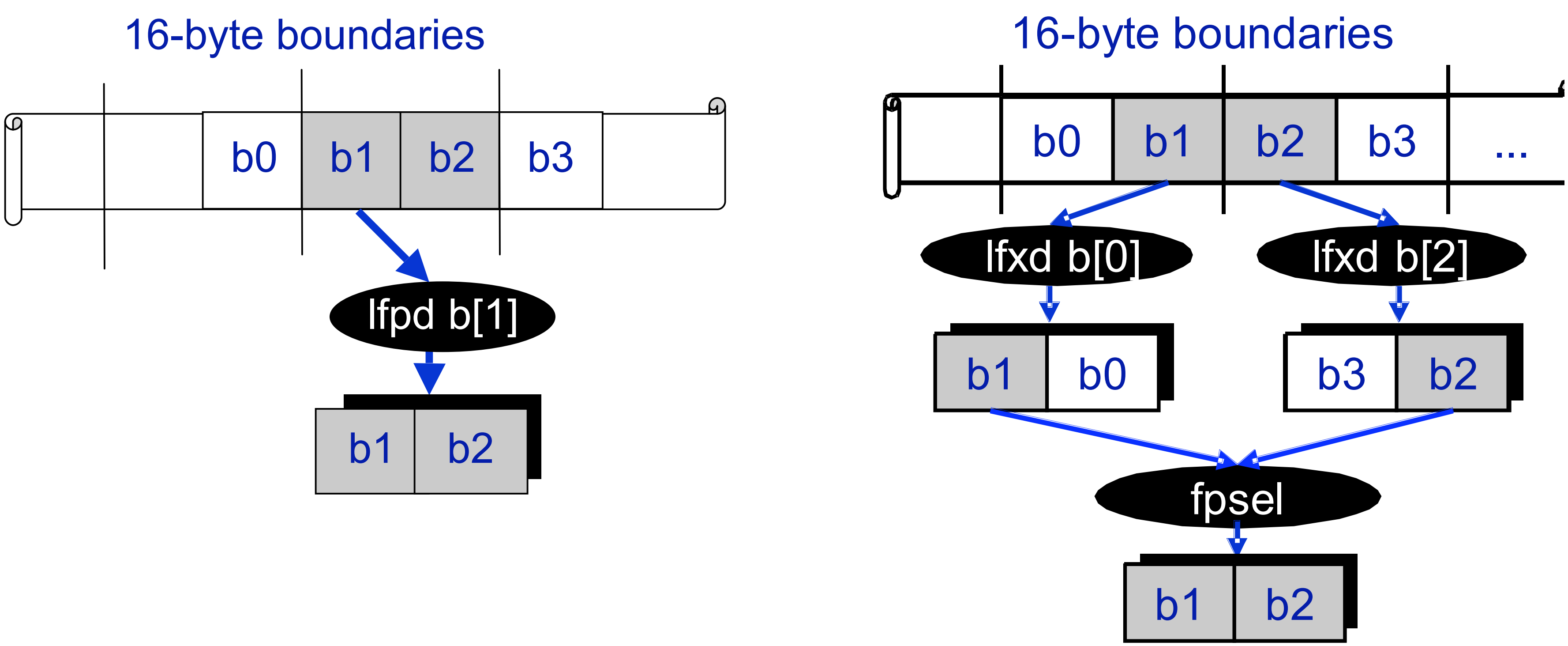


On average: 1 aligned store, 2 aligned cross-loads, 1 add, 1 select

Issues with Runtime Alignment

- Depending on the alignment, different code sequences may be generated
  - When alignment is runtime, the compiler does not know which code sequence tc generate

1. when b[1] is aligned
2. when b[1] is misaligned



Versioning for relative alignment

- Solution to loops with runtime alignment
  - versioning for relative alignment

- When versioning is needed?
  - for (i=0; i<100; i++) a[i+n] = b[i+1+n] + c[i+1+n];
    - n is runtime loop invariant
    - a[i+n], b[i+1+n], c[i+1+n]: runtime alignments, but relatively aligned
    - no versioning is necessary

  - for (i=0; i<100; i++) p[i] = q[i] + r[i];
    - p, q, and r are pointers, alignment & relative alignment unknown
    - versioning is necessary
    - bet on them being relatively aligned

      if ((p-q) mod 16 == 0 && (p-r) mod 16 == 0) $\Rightarrow$ SIMD version

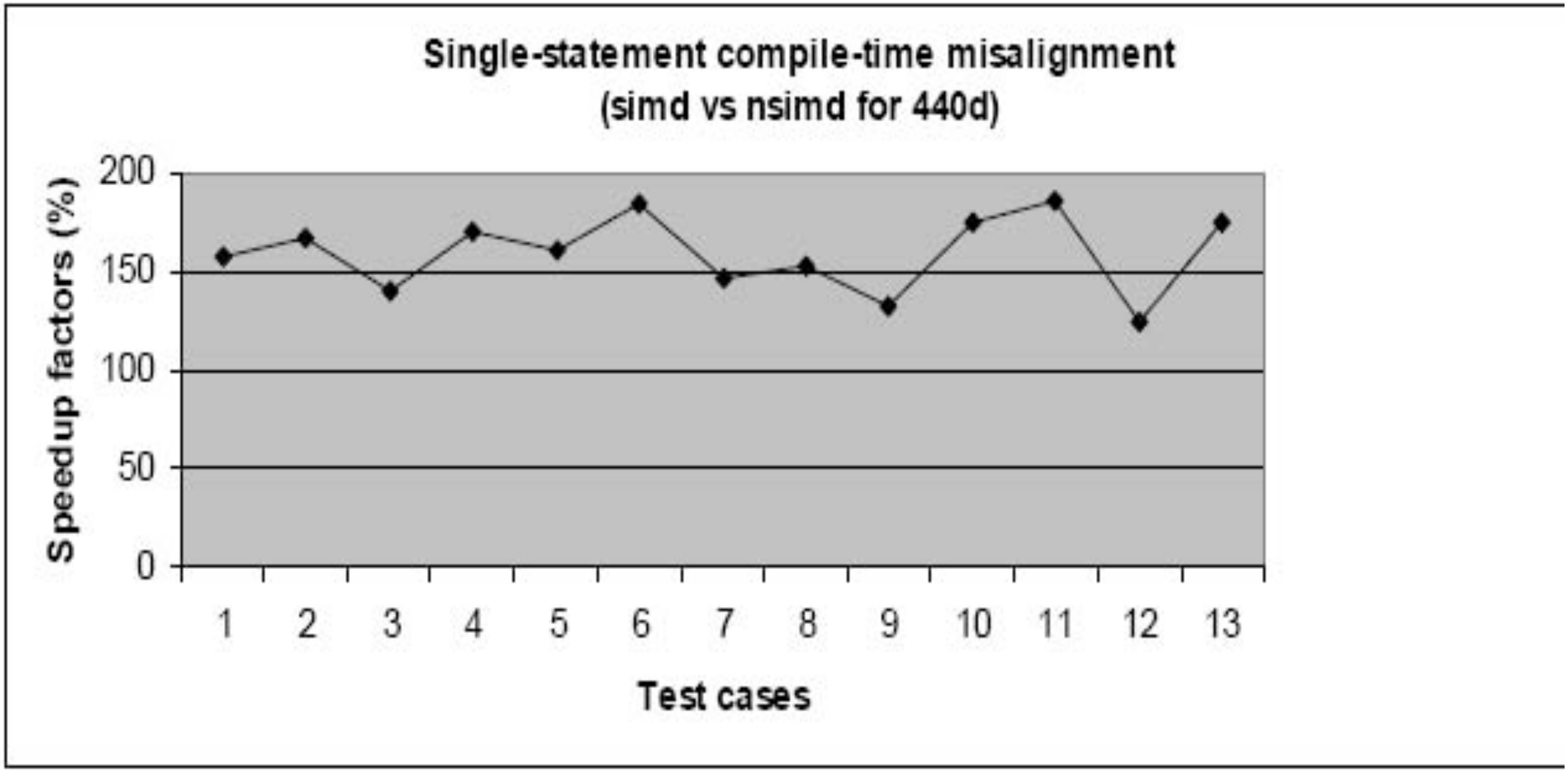
Outline

- Background
- How does the compiler simdize
- Diagnostic info and tuning
- Alignment handling
- Experimental results

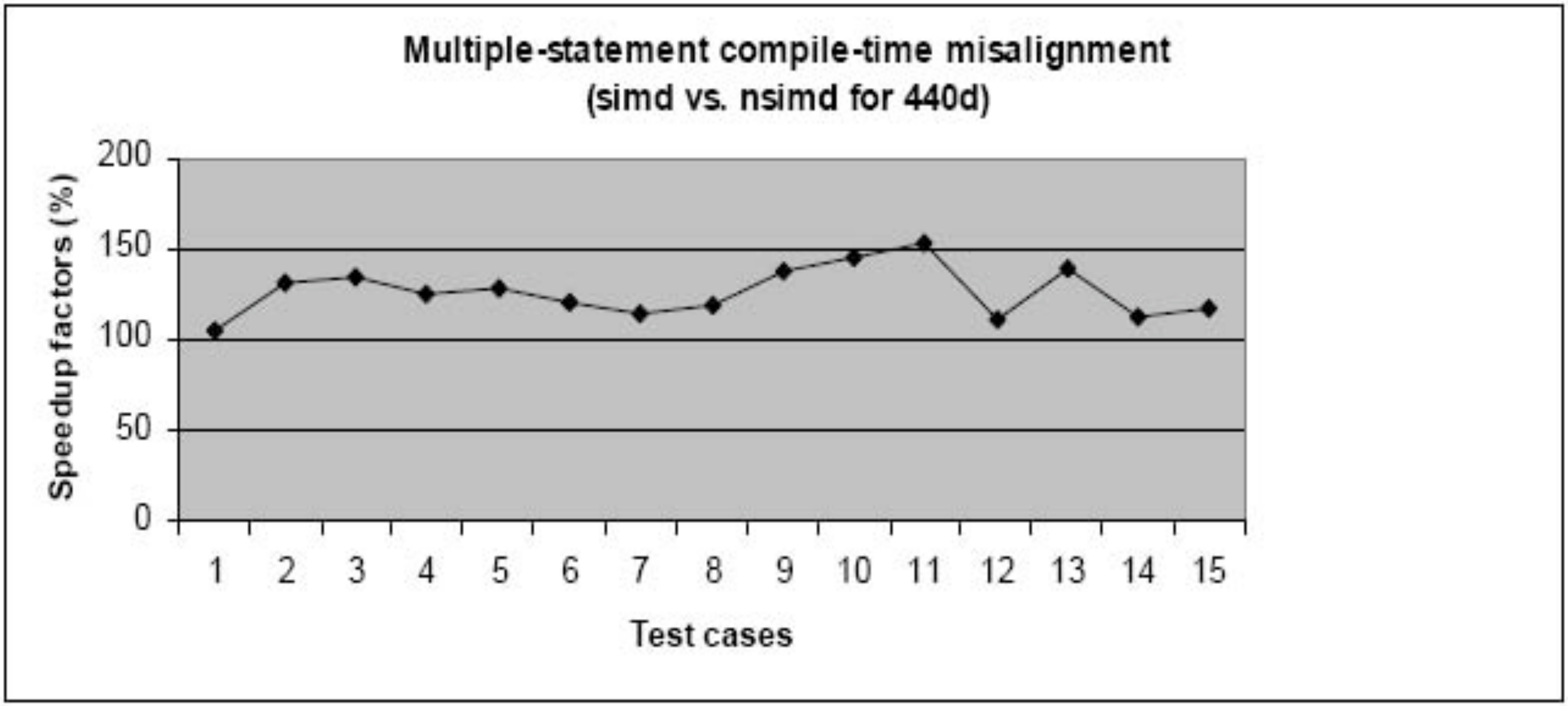
Evaluation of Alignment Handling

- Measurements on a set of kernel loops
  - Harmonic means of a set of 50 loops with identical characteristics
    - 3 loads, 2 adds, 1 store per statement
    - 3 statements per loop for multiple statement loops
    - 500 iterations per loop
    - Randomly generated memory alignments

Single-statement loop with compile-time misalignment

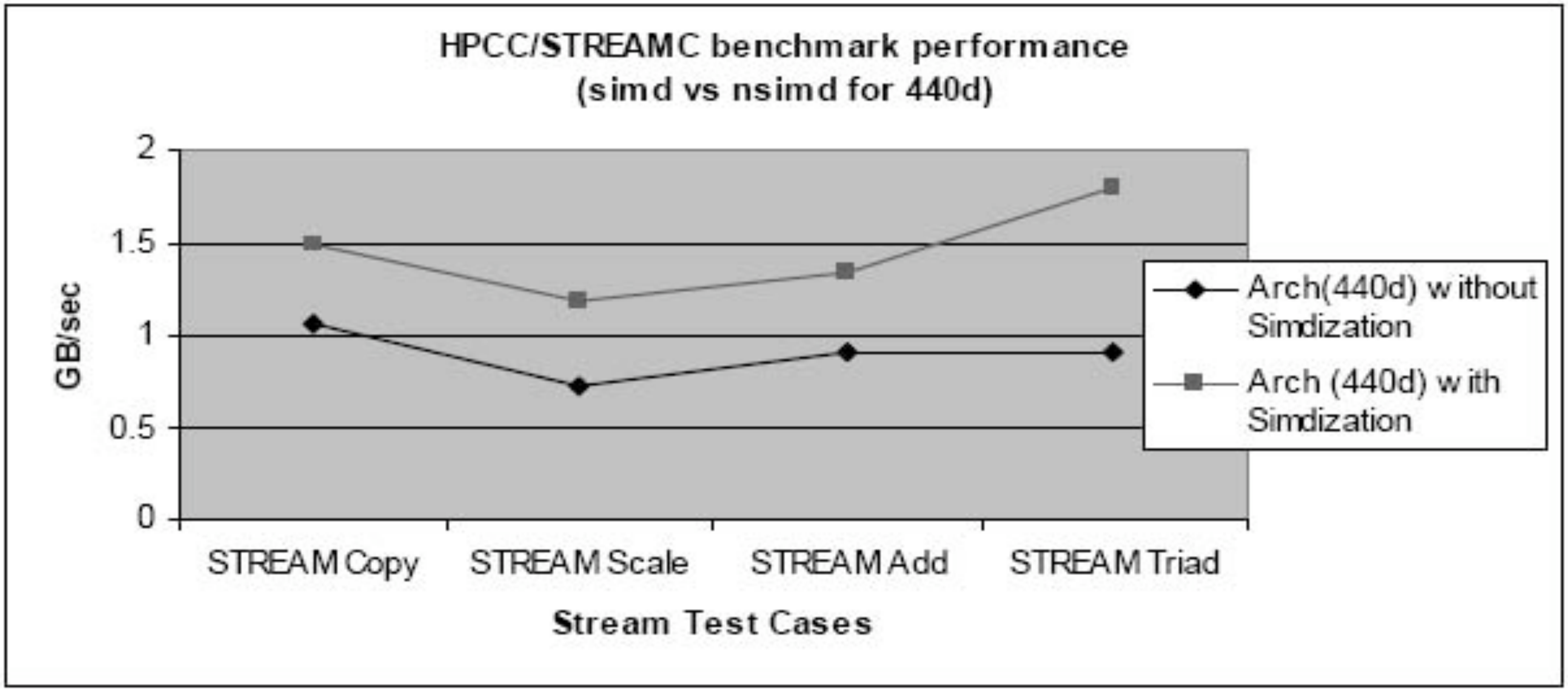


Multiple-statement loops with compile-time misalignment



HPCC/StreamC Simdization performance

- Compiler simdizes all 4 stream tests, speedup factor 1.39 ~ 1.97.



IBM Research

Math Libraries for BlueGene/L

Ramendra K. Sahoo
Blue Gene systems software
IBM Research

Feb. 24, 2005

Outline

- List of Math libraries.
- What the users should keep in mind while choosing math functions
- Benefits from Compiler and how to proceed
- Math library single node performance (Don’t take it seriously!)
- Future Plans

List of Math Libraries

- Engineering & Scientific Subroutine Libraries (ESSL)
  - Only Static libraries
  - No shared libraries
- Mathematics Accelerated Scientific Subroutines (MASS)
- Mathematics Accelerated Scientific Subroutines Vectorized (MASSV)
- FFTW

ESSL for BG/L

- Based on ESSL 4.2 for p-series Linux/AIX.
- Uses same code/ algorithms used for other ESSL releases.
- Core routines optimized to exploit higher order compiler optimizations (-O4, -O5).
- Optimized to provide maximum benefit of 440d (double hummer)
- Always use –qarch=440d, -qtune=440 to get the double hummer code generations..(not necessarily always would provide performance benefits!).

ESSL Modules

	(I)	(S)	(L)
Linear Algebra Subprograms			
Vector-scalar	0	41	41
Sparse vector-scalar	0	11	11
Matrix-vector	1	32	32
Sparse matrix-vector	0	0	3
Matrix Operations			
Addition, subtraction, multiplications, rank-k updates, rank-2k updates and transpose	0	25	26
Linear Algebra Equations			
Dense linear algebraic equations	3	53	58
Banded linear algebraic equations	0	18	18
Sparse linear algebraic equations	0	0	11
Linear least squares	0	3	5

ESSL Modules (Contd..)

	(I)	(S)	(L)
Eigensystem Analysis			
Solutions to general eigensystems & general eigensystem analysis problems	0	8	8
Signal Processing Computations			
Fourier transforms	0	15	11
Convolutions and correlations	0	10	2
Related Computations	0	6	6
Sorting and Searching			
sorting, sorting with index, & binary and sequential searching	5	5	5

ESSL Modules (Contd…)

	(I)	(S)	(L)
Interpolation			
Polynomial and cubic spline interpolation	0	4	4
Numerical Quadrature			
Numeric quadrature on a set of points or on a functions	0	6	6
Random Number Generation			
Generating vectors of uniformly distributed random numbers	0	3	3
Utilities	8	0	3
Total	**13**	**240**	**253**

Planning Your Program

- Select an ESSL subroutine
- Avoid Conflicts with Internal ESSL Routine Names Exported
- Setting up your data
- Setting up your ESSL calling sequences
- Using auxiliary storage in ESSL
- Providing a correct transform length in ESSL
- Getting the best accuracy
- Getting the best performance
- Dealing with errors while using ESSL

Planning Your Program

- An ESSL subroutine is a named sequence of instructions within the ESSL product library whose execution is invoked by a call.

- Interpreting the subroutine names with a prefix underscore

Example :
_GEMUL (all versions of the matrix multiplication subroutine
SGEMUL, DGEMUL, CGEMUL and ZGEMUL

S for short-precision real, D for long-precision real
C for short-precision complex Z for long-precision complex
I for integer

- Syntax

fortran	CALL NAME-1 | NAME-2 | ...| NAME-n (arg-1, arg-2,..., arg-m,..)
C & C++	name-1 | name-2 |....| name-n (arg-1, ..., arg-m,...);

Planning Your Program (Contd..)

- Conflicts with Internal ESSL routines :

  Avoid using **ESV** as prefix names for your names.

- Scalar data passed to ESSL from all types of programs, including Fortran, C,and C++, should confirm to **ANSI/IEEE 32-bit** floating point format as per ANSI/IEEE standard for binary floating-point arithmatic, ANSI/IEEE Standard.

- All arrays, regardless of the type of data, should be aligned to ensure optimal performance. **Alignment exceptions** can be figured out through compilation options.

Planning Your Program (Contd..)

- Conflicts with Internal ESSL routines :

  Avoid using **ESV** as prefix names for your names.

- Scalar data passed to ESSL from all types of programs, including Fortran, C,and C++, should confirm to **ANSI/IEEE 32-bit** floating point format as per ANSI/IEEE standard for binary floating-point arithmatic, ANSI/IEEE Standard.

- All arrays, regardless of the type of data, should be aligned to ensure optimal performance. **Alignment exceptions** can be figured out through compilation options.

ESSL Functional Testing

- We have functional tests carried out with a number of different options
  - -O3 440 (98% success)*.
  - -O3 440d (95% success)*.
  - -O5 440d (91% success)*.

- A number of outstanding defects fixed in compilers (particularly the TPO, TOBEY related).

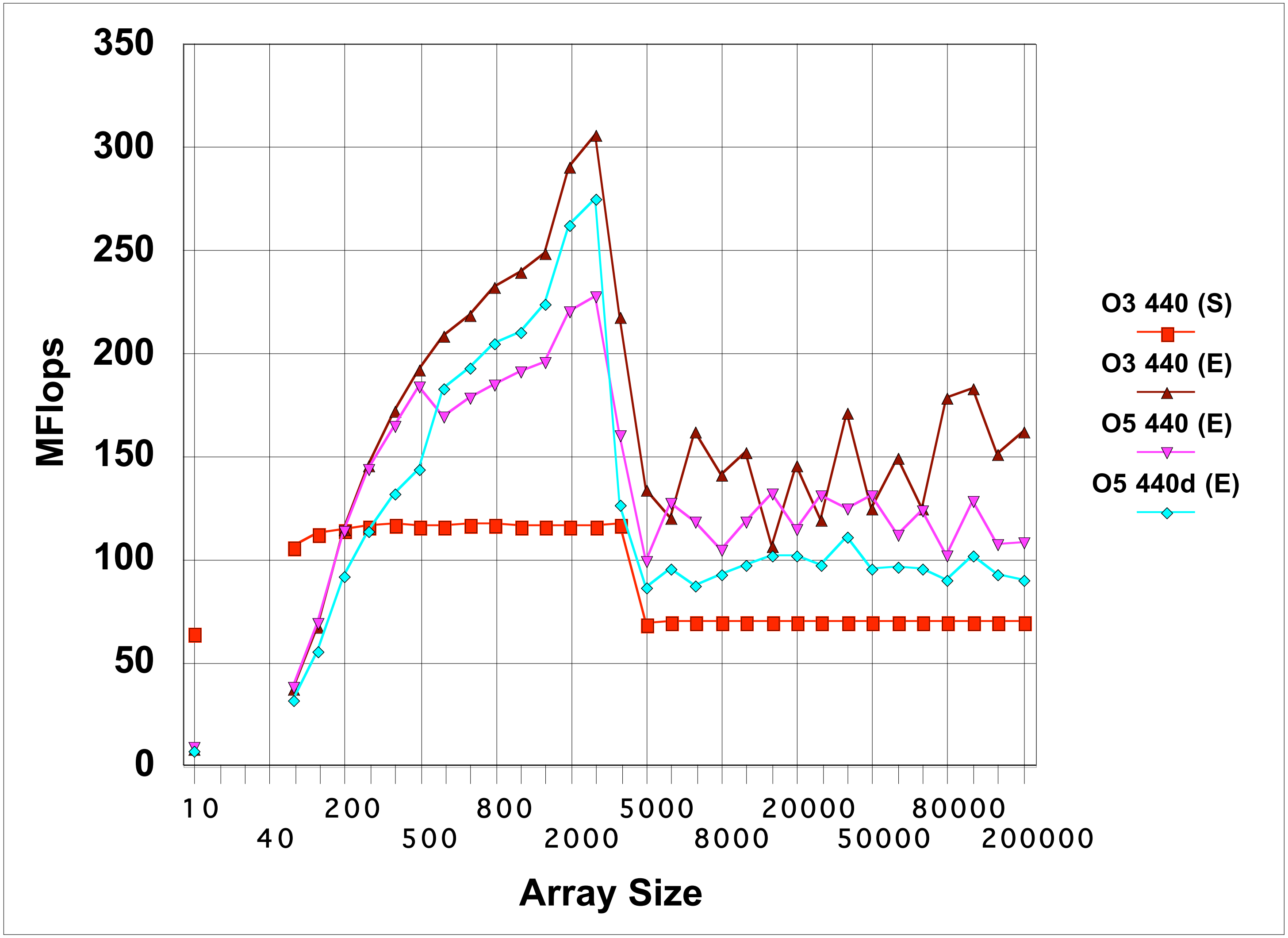
* Based on results for ESSL 4.1 in 2004.

Example Routine (DASUM)

- SUM = DASUM (N,DX,INCX)
- Compute the sum of the absolute values in the vector.
- A comparison of results from vanilla code with ESSL.
- Example codes : dasum.F (removing qstrict, O3)
  dasum_vanilla.F (code from netlib)
  dasum_orig.F (code from ESSL)
- Use –qdebug=diagnostic to examin which loops are simdized.
- **Limitation** : among loops with strides, simdization only possible for stride 1 loops.

  Hence : Vanilla dasum code performance is better than ESSL !
- **Solution** : Adding Pragmas to take care of non stride 1 loops (available in future compiler releases)

ESSL sample performance results



MASS/MASSV Libraries

- Provides elementary math functions in both scalar and vector form
  - Examples : sqrt, pow, inv, log etc.
- Provides trigonometric and hyperbolic math functions in scalar and vector form.
- Examples : sin, cos, tan, atan, sinh

- All the routines are C routines (replacing Assembly routines written for p-series).
- A list of functions already supported :
  - Scalar functions : atan, exp, rsqrt, tanh, sincos, cosh , log sinh sqrt, pow, tan
  - Vector functions : vacos, vcos, vlog1p, vsasin, vsexpm1, vslog, vssinh, vasin, vlog, vsatan2, vsexp, vspow, vssin, vatan2, vdiv, vpow, vscbrt, vsincos, vsqrt, vssqrt, vcbrt, vrcbrt, vscosh, vsinh, vsrcbrt, vstanh, vcosh, vexpm1, vrec, vscosisin, vsin, vsrec, vstan, vexp, vrsqrt, vscos, vslog10, vsrsqrt, vtanh, vcosisin, vlog10, vsacos, vsdiv, vslog1p, vssincos, vtan

Sample MASS library performance benefits

Function	Libmass.a (cycles)	Libm.a (cycles)	
sqrt	42.0	101.0	
rsqrt	35.0	133.0	
exp	56.0	168.0	
log	68.0	316.7	
sin	66.6	191.9	
cos	65.8	199.9	
tan	89.5	316.5	
atan	109.0	216.0	
sinh	81.0	326.1	
cosh	68.0	239.4	
pow	157.0	521.3	

Summary

- We provide the same set of math libraries for BG/L as provided in other IBM platforms .
- Functionality part has been tested and verified.(2004)
- Math libraries will have significant performance improvements in next few months.(Specifically ESSL release for BG/L targetted for October 2005)
- Higher optimizations (-O4/ –O5) a reality !
- A number of compiler fixes and improvements (including special pragmas supporting math functions).
- Need your feed back in terms of performance/tuning results to further improve.

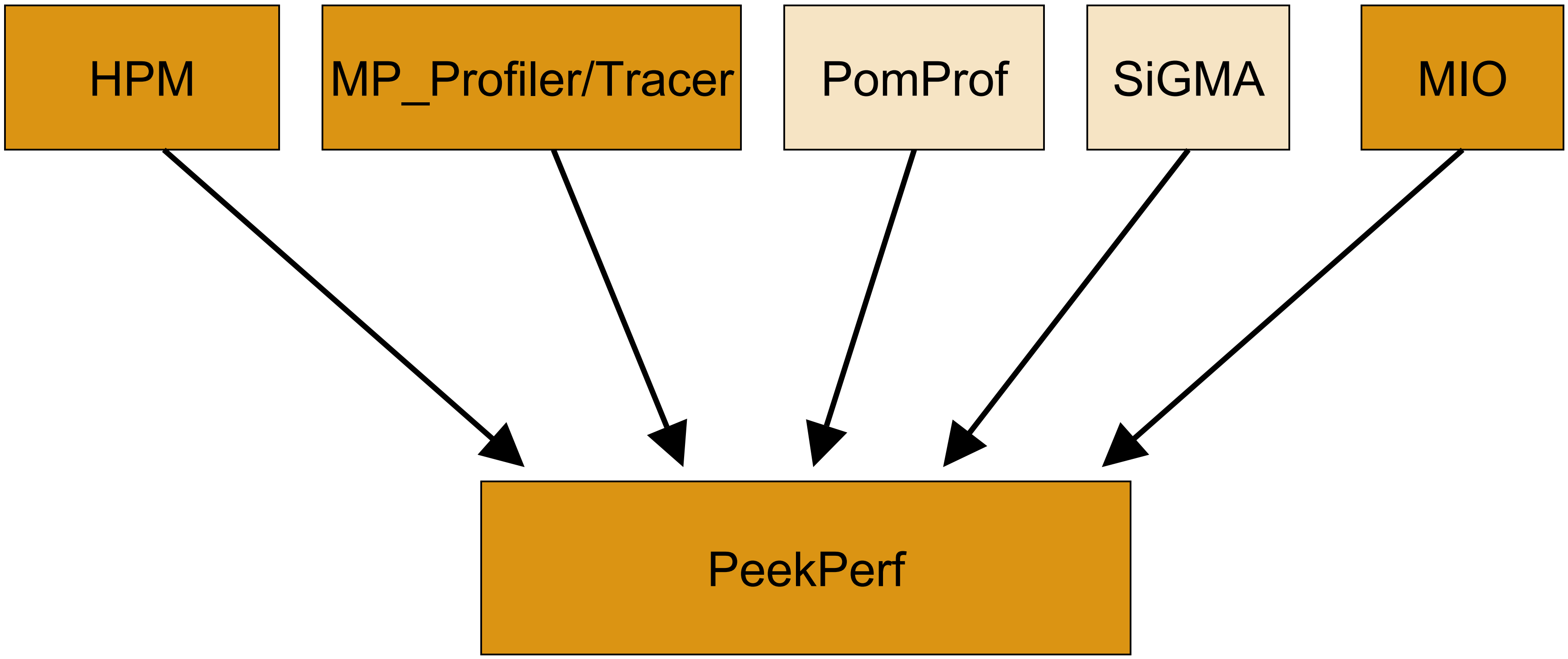
The IBM High Performance Computing Toolkit on BlueGene/L

I-Hsin Chung ihchung@us.ibm.com
Guojing Cong gcong@us.ibm.com
David Klepacki klepacki@us.ibm.com

Outline

- **IBM High Performance Computing Toolkit**
  - MPI performance: MP_Profiler
  - CPU performance: Xprofiler, HPM
  - Modular I/O: MIO
  - Visualization and analysis: PeekPerf
- **Related Tools**
- **Challenges**

IBM HPCT Overview



Message-Passing Performance:

- **MP_Profiler Library**
  - Captures “summary” data for MPI calls
  - Source code traceback
  - User **MUST** call MPI_Finalize() in order to get output files.
  - No changes to source code
    - MUST compile with –g to obtain source line number information
- **MP_Tracer Library**
  - Captures “timestamped” data for MPI calls
  - Source traceback

MP_Profiler Summary Output

MPI Routine	#calls	avg. bytes	time(sec)
MPI_Comm_size	3	0.0	0.000
MPI_Comm_rank	12994	0.0	0.016
MPI_Send	19575	11166.9	13.490
MPI_Isend	910791	5804.2	9.216
MPI_Recv	138173	2767.9	73.835
MPI_Irecv	784936	15891.6	2.407
MPI_Sendrecv	894809	352.0	88.705
MPI_Wait	1537375	0.0	288.049
MPI_Waitall	44042	0.0	25.312
MPI_Bcast	464	41936.8	3.272
MPI_Barrier	1312	0.0	34.206
MPI_Gather	68	16399.1	2.680
MPI_Scatter	6	17237.3	0.532

```
total communication time = 770.424 seconds.
total elapsed time       = 1168.662 seconds.
user cpu time            = 1160.960 seconds.
system time              = 0.620 seconds.
maximum memory size      = 68364 KBytes.

To check load balance :  grep "total comm" mpi_profile.*
```

MP_Profiler Sample Call Graph Output

```
communication time = 143.940 sec, parent = gwwrloc
   MPI Routine              #calls        time(sec)
   MPI_Barrier                2311          143.734
   MPI_Gatherv                2311            0.206
communication time = 137.823 sec, parent = f2drecv
   MPI Routine              #calls        time(sec)
   MPI_Recv                  91959          137.823
communication time = 108.960 sec, parent = puttsf
   MPI Routine              #calls        time(sec)
   MPI_Barrier               23607          106.821
   MPI_Gatherv               23607            2.139
communication time = 94.435 sec, parent = fft_tri_recv
   MPI Routine              #calls        time(sec)
   MPI_Recv                   6378           94.435
communication time = 83.836 sec, parent = fft2drecv
   MPI Routine              #calls        time(sec)
   MPI_Recv                  93003           83.836
```

MP_Profiler Message Size Distribution

```
msglen =       8 bytes,   elapsed time = 0.0029 msec
msglen =       8 bytes,   elapsed time = 0.0029 msec
msglen =      32 bytes,   elapsed time = 0.0030 msec
msglen =      64 bytes,   elapsed time = 0.0028 msec
msglen =      96 bytes,   elapsed time = 0.0028 msec
msglen =     160 bytes,   elapsed time = 0.0029 msec
msglen =     240 bytes,   elapsed time = 0.0030 msec
msglen =     320 bytes,   elapsed time = 0.0030 msec
msglen =     400 bytes,   elapsed time = 0.0030 msec
msglen =     480 bytes,   elapsed time = 0.0031 msec
                     . . .
                     . . .
                     . . .
msglen =  640000 bytes,   elapsed time = 0.3616 msec
msglen =  720000 bytes,   elapsed time = 0.4019 msec
msglen =  800000 bytes,   elapsed time = 0.4630 msec
msglen = 1000000 bytes,   elapsed time = 0.6618 msec
```

Hardware Performance Monitor (HPM)

- **Provides comprehensive reports of events that are critical to performance on IBM systems.**
- **Gather critical hardware performance metrics, e.g.**
  - Number of misses on all cache levels
  - Number of floating point instructions executed
  - Number of instruction loads that cause TLB misses
- **Helps to identify and eliminate performance bottlenecks.**

HPM

- **hpmcount**
  - Starts application and provides
    - Wall clock time
    - Hardware performance counter information
    - Resource utilization statistics
- **libhpm**
  - Library for program (including multi-thread) section instrumentation.
- **hpmstat**
  - Provides system wide reports for root

Hpmcount Example

```
bash-2.05a$ hpm_count swim
……  // program output
 hpmcount (V 2.5.4) summary
 Execution time (wall clock time)                          : 7.378159 seconds
 ########  Resource Usage Statistics  ########
 Total amount of time in user mode                         : 0.010000 seconds
Average shared memory use in text segment                  : 672 Kbytes*sec
………
  PM_FPU_FDIV (FPU executed FDIV instruction)              : 0
  PM_FPU_FMA (FPU executed multiply-add instruction)       : 0
  PM_CYC (Processor cycles)                                : 54331072
  PM_FPU_STF (FPU executed store instruction)              : 2172
  PM_INST_CMPL (Instructions completed)                    : 17928229
  Utilization rate                                         : 0.446 %
  Total load and store operations                          : 0.004 M
  MIPS                                                     : 2.140
  Instructions per cycle                                   : 0.330
```

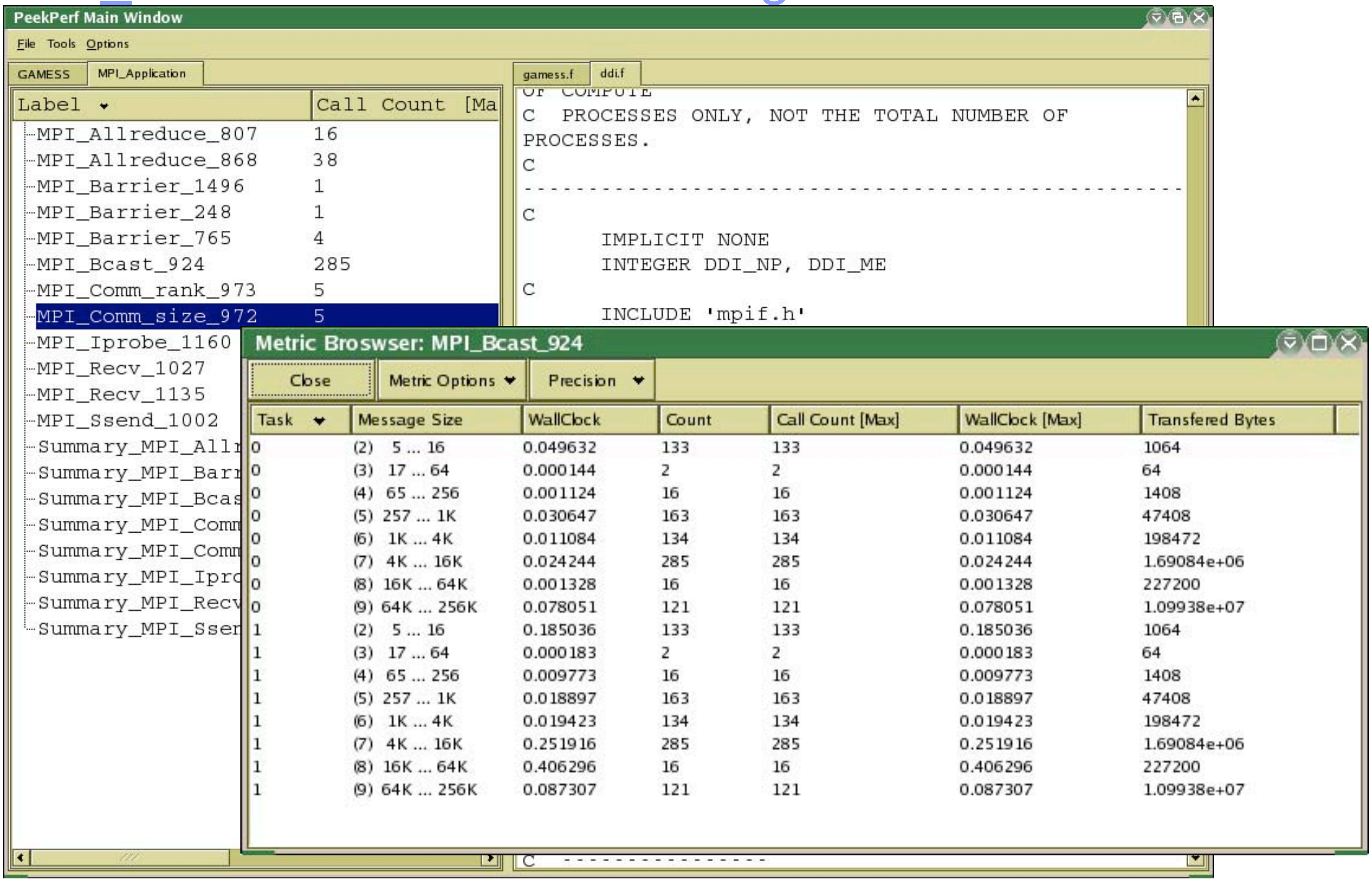
Libhpm Example

```
 call f_hpmstart( 30, "Loop 300" )
C$OMP PARALLELDO
C$OMP&SHARED (ALPHA,M,N,U,V,P,UNEW,VNEW,PNEW,UOLD,VOLD,POLD)
C$OMP&SHARED (JS,JE)
C$OMP&PRIVATE (I,J)
      DO 300 J=js,je
      DO 300 I=1,M
      UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-2.*U(I,J)+UOLD(I,J))
       ……
       P(I,J) = PNEW(I,J)
 300 CONTINUE
      call f_hpmstop( 30 )
```

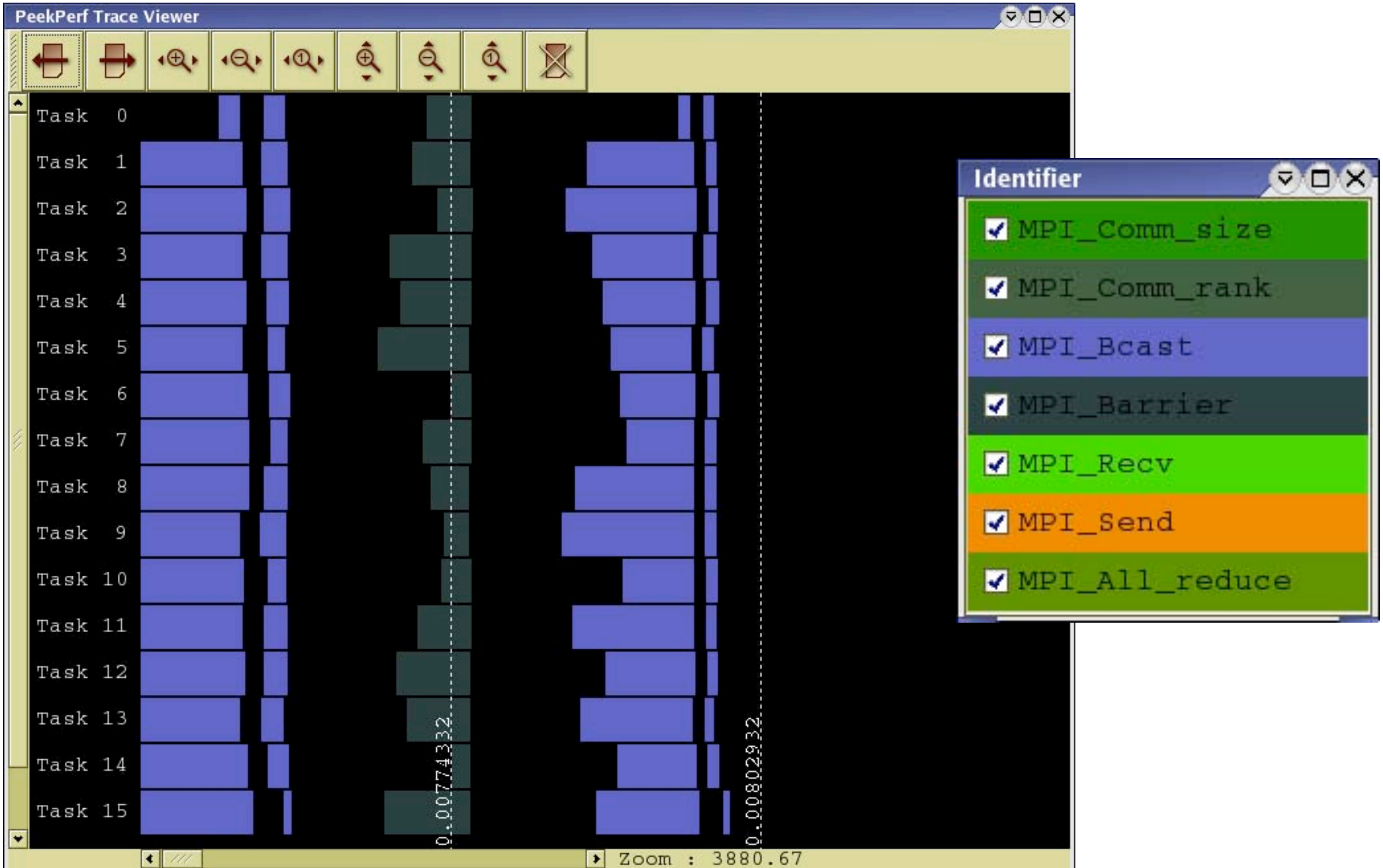
Modular I/O (MIO)

- **Addresses the need of application-level optimization for I/O.**
- **Analyze and tune I/O at the application level**
  - For example, when an application exhibits the I/O pattern of sequential reading of large files
  - MIO
    - Detects the behavior
    - Invokes its asynchronous prefetching module to prefetch user data.
- **Planned Integration into HPC Toolkit with PeekPerf capabilities**
  - Source code traceback
  - Future capability for dynamic I/O instrumentation

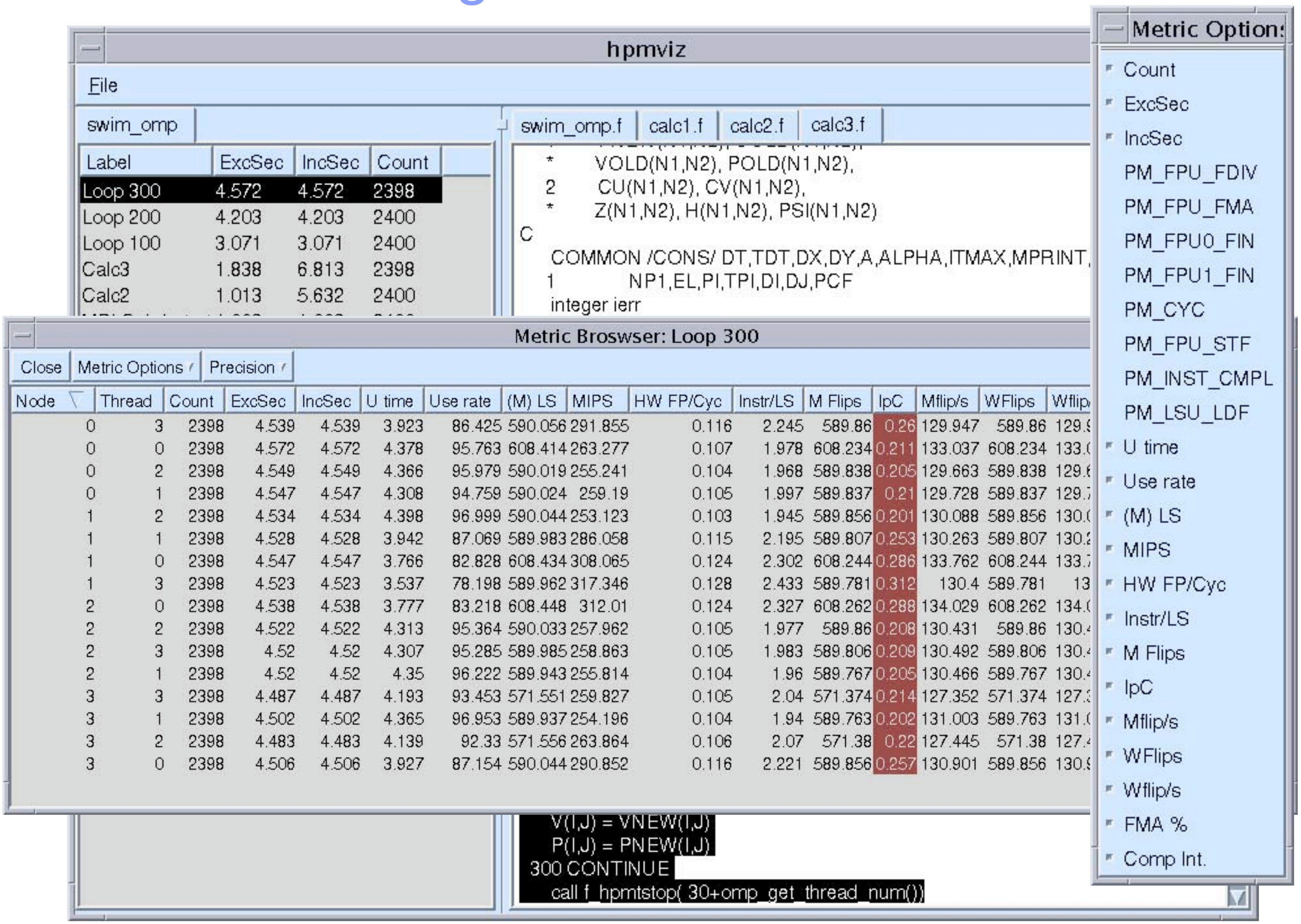
MP_Profiler Visualization Using PeekPerf



MP_Tracer Visualization Using PeekPerf

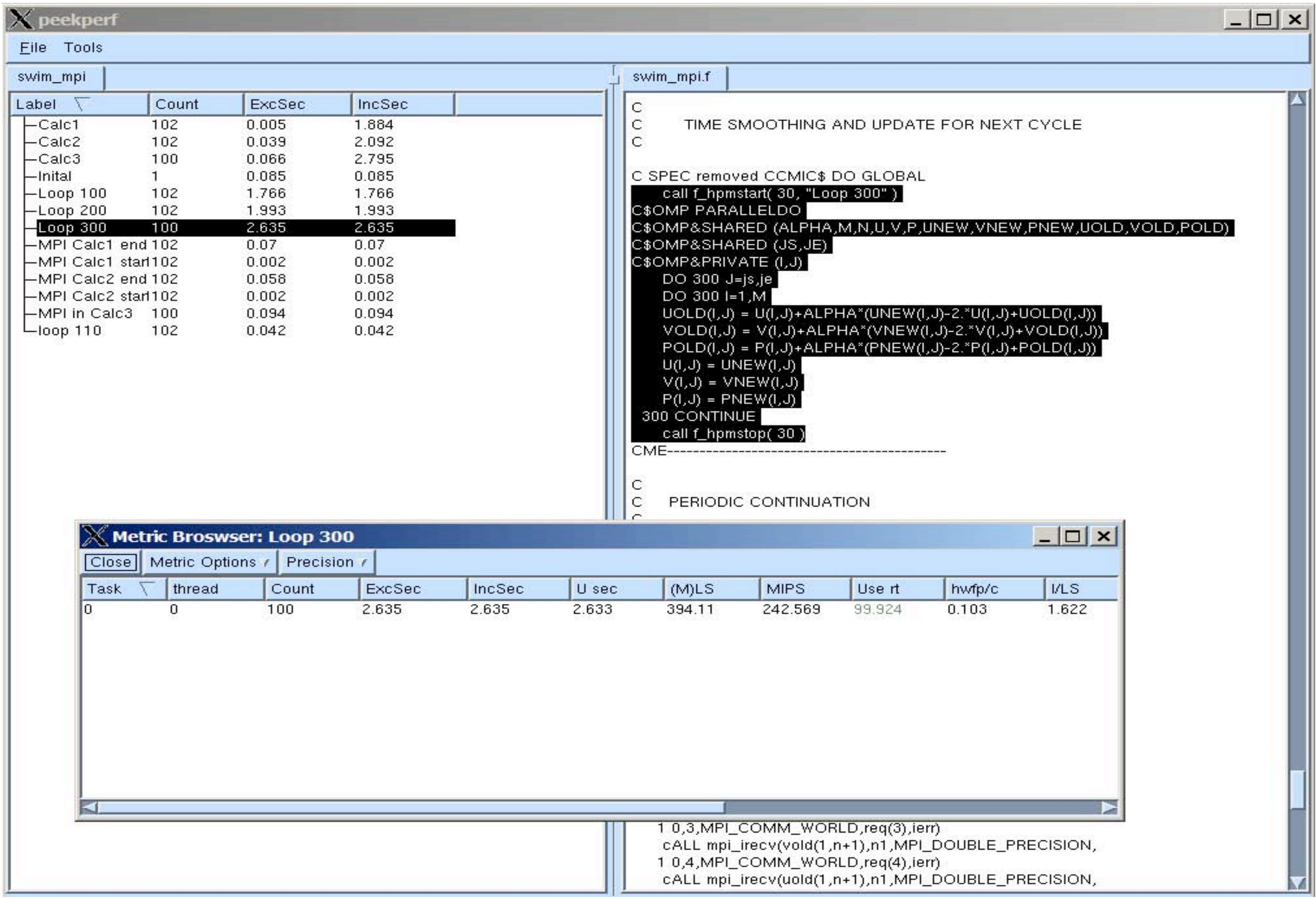


HPM Visualization Using PeekPerf



Node	Thread	Count	ExcSec	IncSec	U time	Use rate	(M) LS	MIPS	HW FP/Cyc	Instr/LS	M Flips	IpC	Mflip/s	WFlips	Wflip
0	3	2398	4.539	4.539	3.923	86.425	590.056	291.855	0.116	2.245	589.86	0.26	129.947	589.86	129.
0	0	2398	4.572	4.572	4.378	95.763	608.414	263.277	0.107	1.978	608.234	0.211	133.037	608.234	133.
0	2	2398	4.549	4.549	4.366	95.979	590.019	255.241	0.104	1.968	589.838	0.205	129.663	589.838	129.
0	1	2398	4.547	4.547	4.308	94.759	590.024	259.19	0.105	1.997	589.837	0.21	129.728	589.837	129.
1	2	2398	4.534	4.534	4.398	96.999	590.044	253.123	0.103	1.945	589.856	0.201	130.088	589.856	130.
1	1	2398	4.528	4.528	3.942	87.069	589.983	286.058	0.115	2.195	589.807	0.253	130.263	589.807	130.
1	0	2398	4.547	4.547	3.766	82.828	608.434	308.065	0.124	2.302	608.244	0.286	133.762	608.244	133.
1	3	2398	4.523	4.523	3.537	78.198	589.962	317.346	0.128	2.433	589.781	0.312	130.4	589.781	13
2	0	2398	4.538	4.538	3.777	83.218	608.448	312.01	0.124	2.327	608.262	0.288	134.029	608.262	134.
2	2	2398	4.522	4.522	4.313	95.364	590.033	257.962	0.105	1.977	589.86	0.208	130.431	589.86	130.
2	3	2398	4.52	4.52	4.307	95.285	589.985	258.863	0.105	1.983	589.806	0.209	130.492	589.806	130.
2	1	2398	4.52	4.52	4.35	96.222	589.943	255.814	0.104	1.96	589.767	0.205	130.466	589.767	130.
3	3	2398	4.487	4.487	4.193	93.453	571.551	259.827	0.105	2.04	571.374	0.214	127.352	571.374	127.
3	1	2398	4.502	4.502	4.365	96.953	589.937	254.196	0.104	1.94	589.763	0.202	131.003	589.763	131.
3	2	2398	4.483	4.483	4.139	92.33	571.556	263.864	0.106	2.07	571.38	0.22	127.445	571.38	127.
3	0	2398	4.506	4.506	3.927	87.154	590.044	290.852	0.116	2.221	589.856	0.257	130.901	589.856	130.

Check Performance Counter



Xprofiler

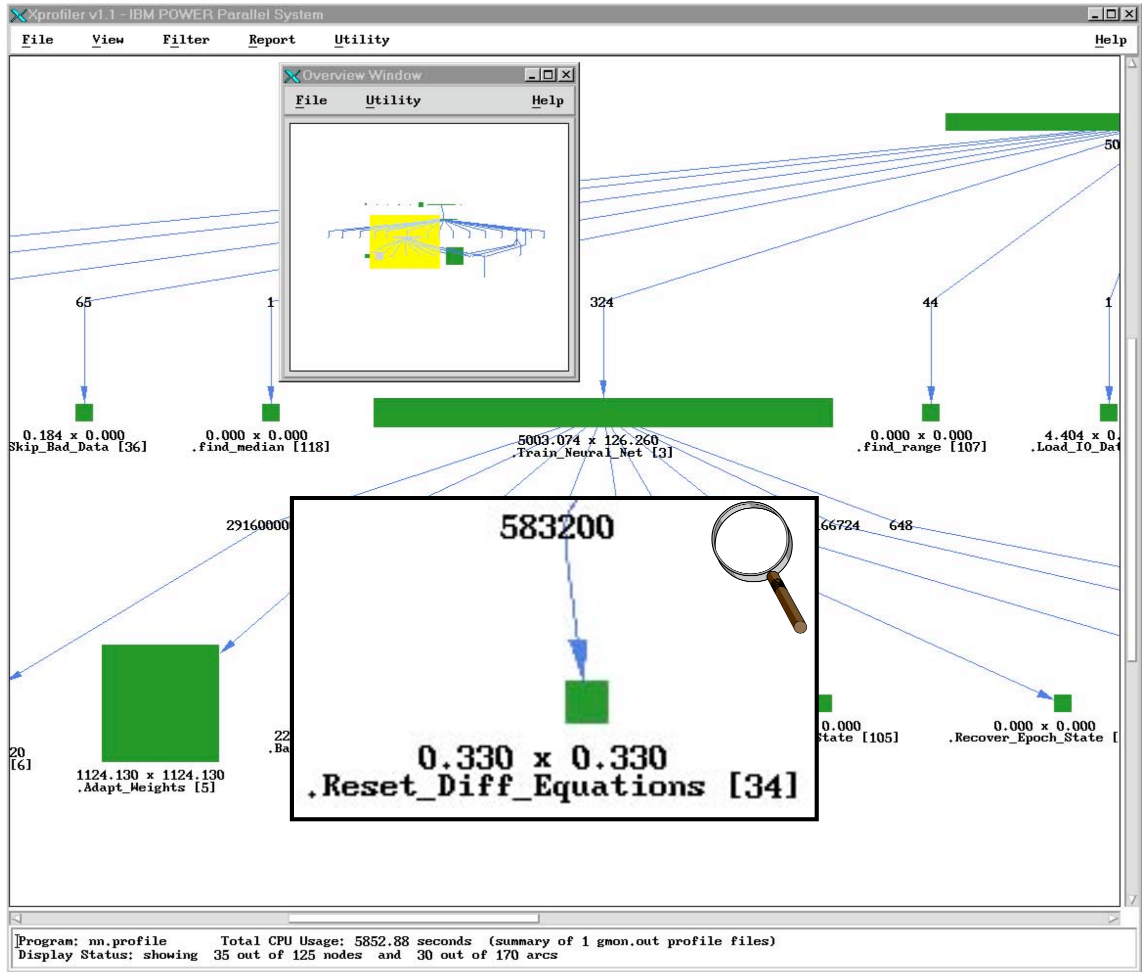
- **CPU profiling tool, extension of gprof**
  - can be used to profile both serial and parallel applications.
- **Graphical display of the call graphs of the application**
- **Provides quick access to the profiled data**
- **Helps users identify the CPU-intensive functions**

Running Xprofiler

- **Compile the program with –pg**
- **Run the program**
- **gmon.out file is generated (MPI applications generate gmon.out.1, …, gmon.out.n)**
- **Run Xprofiler**

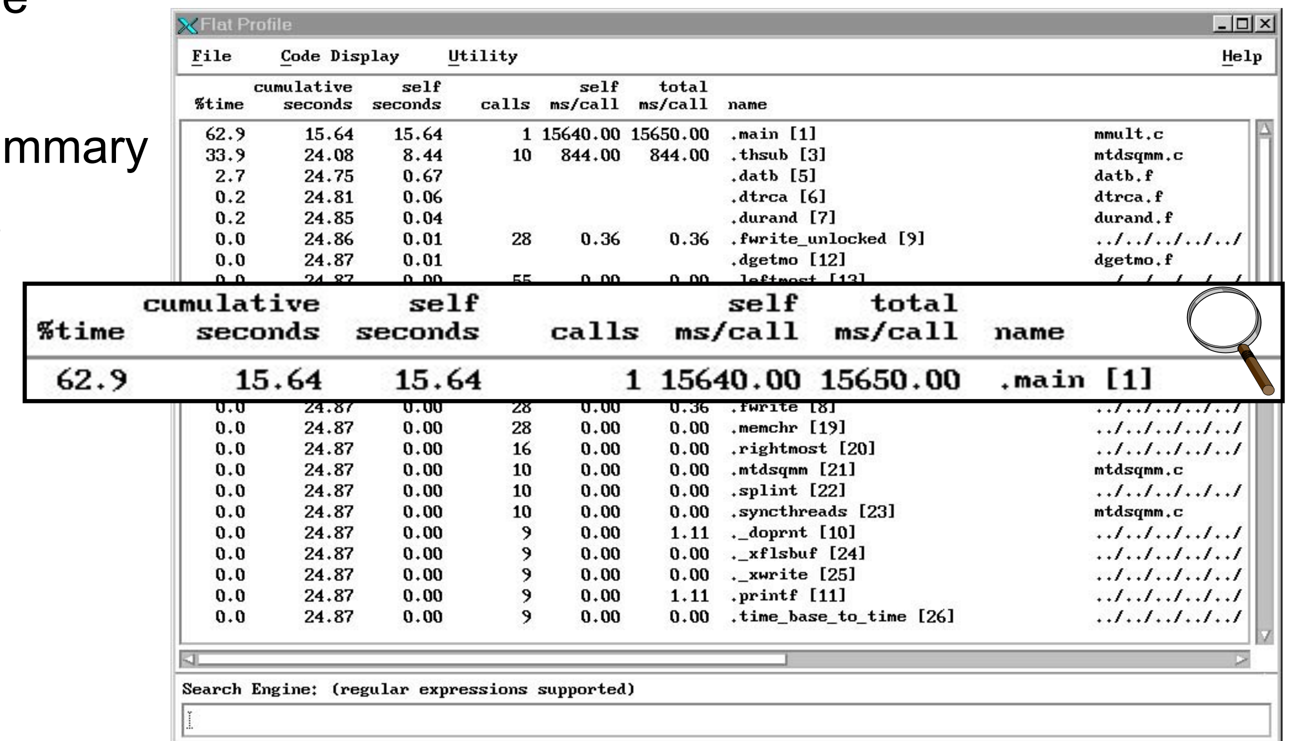
Xprofiler: Main Display

- Width of a bar: time including called routines
- Height of a bar: time excluding called routines
- Call arrows labeled with number of calls
- Overview window for easy navigation (View → Overview)



Xprofiler: Flat Profile

- **Menu `Report` provides usual gprof reports plus some extra ones**
  - Flat Profile
  - Call Graph Profile
  - Function Index
  - Function Call Summary
  - Library Statistics



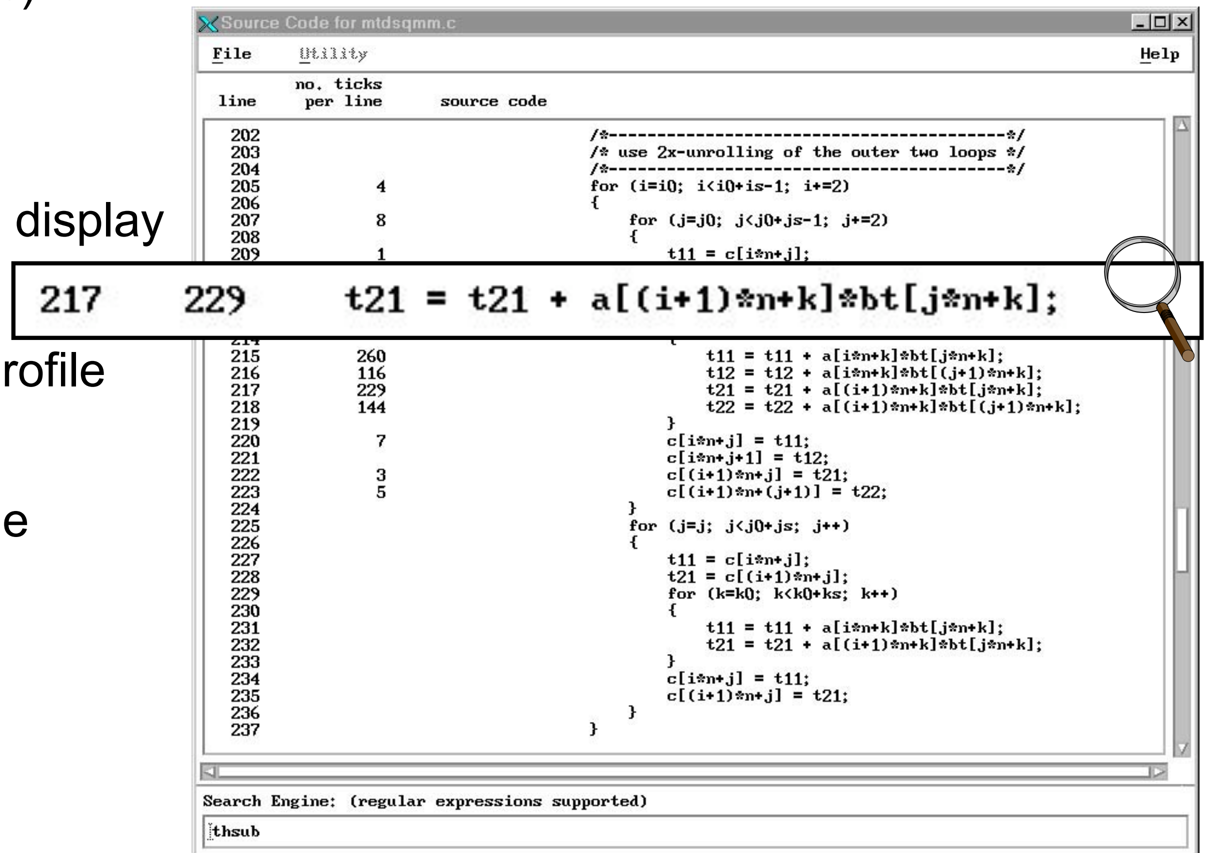
%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name	
62.9	15.64	15.64	1	15640.00	15650.00	.main [1]	mmult.c
33.9	24.08	8.44	10	844.00	844.00	.thsub [3]	mtdsqmm.c
2.7	24.75	0.67				.datb [5]	datb.f
0.2	24.81	0.06				.dtrca [6]	dtrca.f
0.2	24.85	0.04				.durand [7]	durand.f
0.0	24.86	0.01	28	0.36	0.36	.fwrite_unlocked [9]	../../../../../
0.0	24.87	0.01				.dgetmo [12]	dgetmo.f
0.0	24.87	0.00	28	0.00	0.00	.memchr [19]	../../../../../
0.0	24.87	0.00	16	0.00	0.00	.rightmost [20]	../../../../../
0.0	24.87	0.00	10	0.00	0.00	.mtdsqmm [21]	mtdsqmm.c
0.0	24.87	0.00	10	0.00	0.00	.splint [22]	../../../../../
0.0	24.87	0.00	10	0.00	0.00	.syncthreads [23]	mtdsqmm.c
0.0	24.87	0.00	9	0.00	1.11	._doprnt [10]	../../../../../
0.0	24.87	0.00	9	0.00	0.00	._xflsbuf [24]	../../../../../
0.0	24.87	0.00	9	0.00	0.00	._xwrite [25]	../../../../../
0.0	24.87	0.00	9	0.00	1.11	.printf [11]	../../../../../
0.0	24.87	0.00	9	0.00	0.00	.time_base_to_time [26]	../../../../../

%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
62.9	15.64	15.64	1	15640.00	15650.00	.main [1]

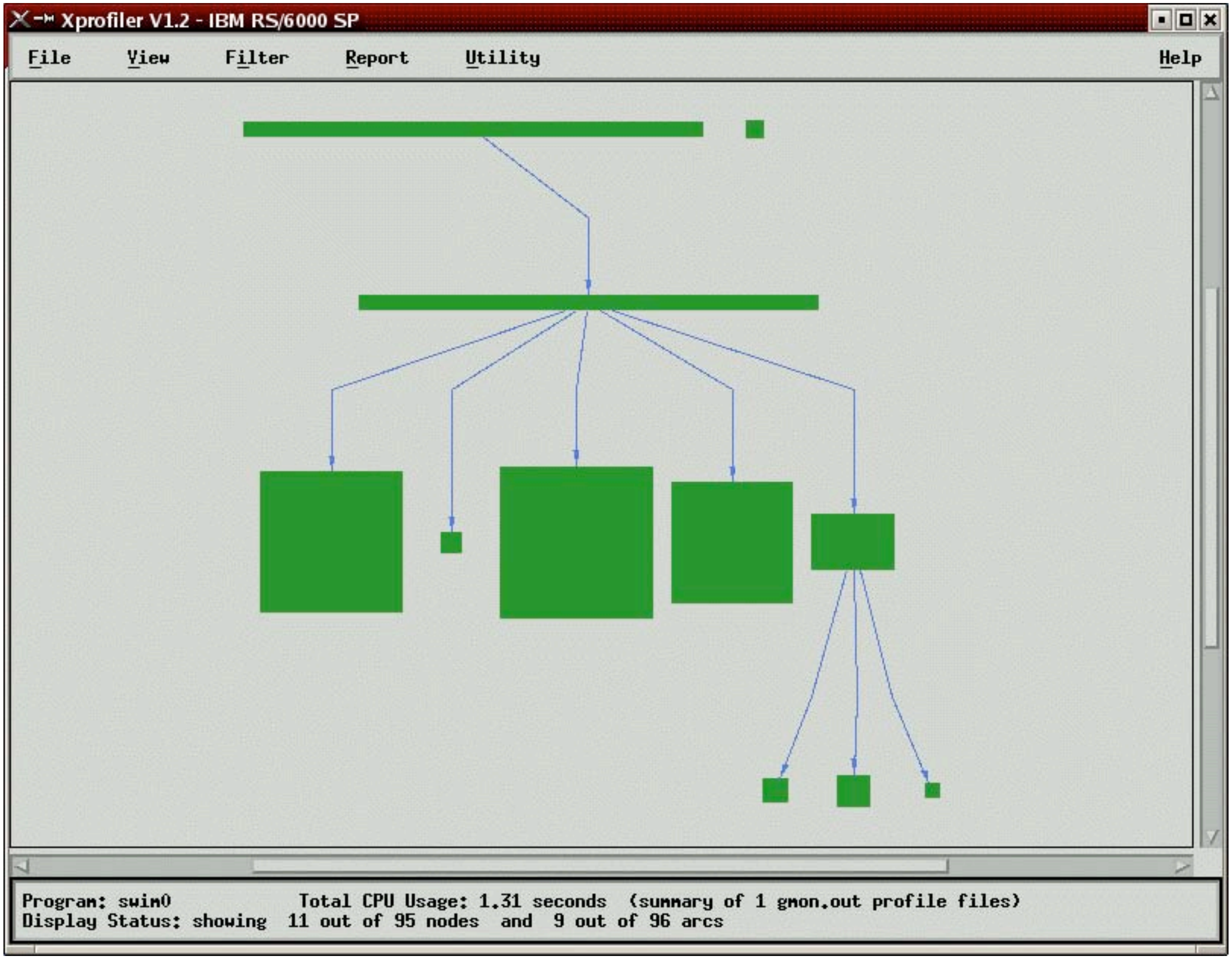
Xprofiler: Source Code Window

- Source code Window displays source code with time profile (resolution 1 tick = 0.01 sec)

- Access
  - Select function in main display
  - → context menu
  - Select function in flat profile
  - → Code Display
  - → Show Source Code



Xprofiler - Application View



Related Work

- **PARAVER (UPC)**
  - Performance Visualization and Analysis Tool
  - Flexibility to represent traces from different environment
  - MPI trace, HW performance counter info (based on PAPI)
- **PAPI (Univ. of Tennessee)**
  - **P**erformance **A**pplication **P**rogramming **I**nterface
  - Design, standardize, and implement a portable and efficient API to access the hardware performance counters

Related Work (continued)

- **TAU (Univ. of Oregon)**
  - **T**uning and **A**nalysis **U**tilities
  - Program and performance analysis tool framework for high-performance parallel and distributed computing
- **mpiP (LLNL/ORNL)**
  - Lightweight profiling library for MPI applications
  - Only collects statistical information about MPI functions
  - With less overhead and have much less data than tracing tools
  - Only uses communication during report generation stage

Challenges

- **Hardware**
  - Different performance counter set
- **Scalability**
  - 64k nodes
  - Tracking communications for each pair:
    - $(65{,}536)^2$x48 bytes = 200 GB
  - Number of processes and events vs. number of screen pixels

Scalability

- **Abstraction**
  - Aggregated performance data
  - Sampling
  - Easy to manage
  - May lose fidelity
- **Performance data organization**
  - Database with one time processing
  - Query, data mining, clustering… etc.
  - Precise information
  - Hard to analyze

Summary

- **IBM High Performance Computing Toolkit**
  - Ported
    - PeekPerf, MP_Profiler
  - Work in progress
    - Xprofiler, HPM
  - Next target
    - Modular I/O: MIO
- **Related Tools**
- **Challenges**
  - Architecture, Scalability

DPOMP

- **Dynamically instruments OpenMP applications**
- **Reports various performance related information**
  - timings, overhead of OpenMP constructs.
- **Modify binaries with performance instrumentation**
  - without requiring access to source codes or recompilation.
- **POMP (Performance monitoring interface for OpenMP) implementation based on dynamic probes**

Simulation Guided Memory Analyzer (SiGMA)

- **Helps to understand precise memory references causing poor memory utilization**
- **Provides fine-grained information useful for**
  - Tuning loop kernels, understanding the cache behavior
- **Consists of**
  - A pre-execution tool that instruments all instructions that refer to memory locations,
  - A runtime data collection tool that performs compression of the stream of memory addresses generated by the instrumentation,
  - Analysis tools that process memory reference trace to provide programmers with tuning information.

Benchmarks on BG/L: Parallel and Serial

John A. Gunnels
Mathematical Sciences Dept.
IBM T. J. Watson Research Center

Overview

- Single node experience
  - Architectural impact
  - Algorithms
- Linpack
  - Dealing with a bottleneck
  - Communication operations

Compute Node: BG/L

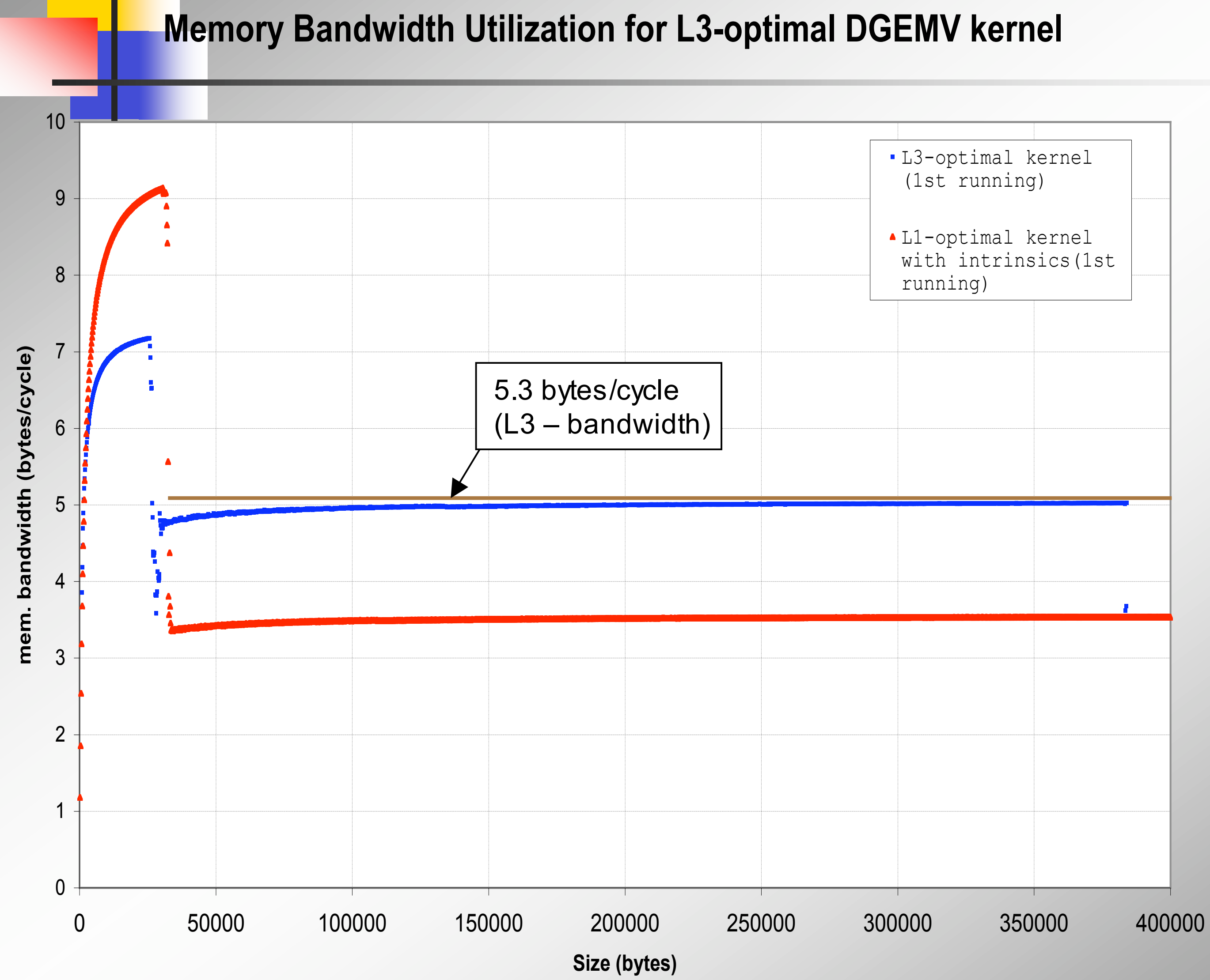
- Dual FPU/SIMD
  - Alignment issues
- Three-level cache
  - Pre-fetching
- Dual Core
- Non-coherent L1 caches
  - 32 KB, 64-way, Round-Robin
  - L2 & L3 caches coherent

Programming Options

High → Low Level

- Compiler optimization to find SIMD parallelism
  - User input for specifying memory alignment and lack of aliasing
    - `alignx` assertion
    - `disjoint` pragma
- Dual FPU intrinsics ("built-ins")
  - Complex data type used to model pair of double-precision numbers that occupy a (P, S) register pair
  - Compiler responsible for register allocation and scheduling
- In-line assembly
  - User responsible for instruction selection, register allocation, and scheduling

L1 and L3-optimal DGEMV Bandwidth Utilization



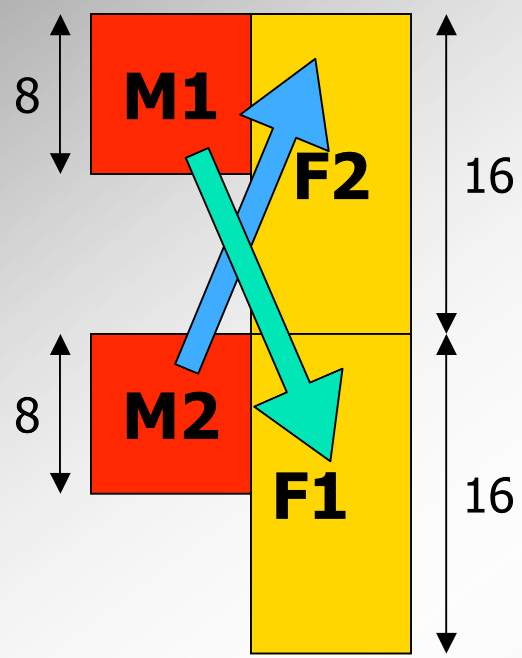
Two different kernels are needed to deal with data when:

-Data come out of L1

-Data come out of L3

Matrix Multiplication

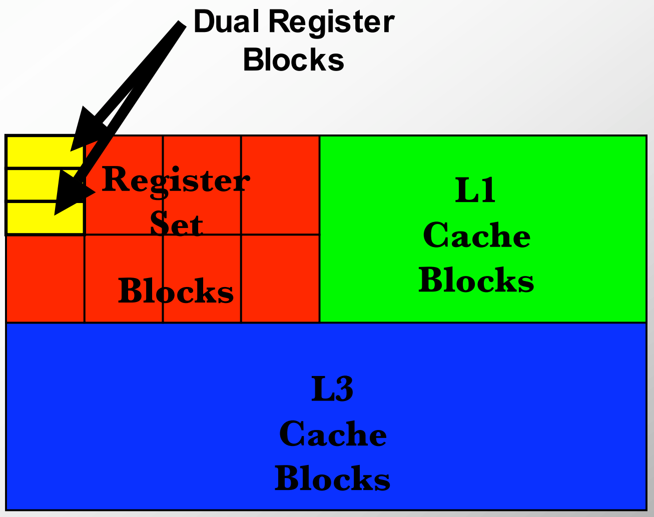
Tiling for Registers (Analysis)



- Latency tolerance (not bandwidth)
  - Take advantage of register count
- Unroll by factor of two
  - 24 register pairs
  - 32 cycles per unrolled iteration
  - 15 cycle load-to-use latency (L2 hit)
- Could go to 3-way unroll if needed
  - 32 register pairs
  - 32 cycles per unrolled iteration
  - 31 cycle load-to-use latency

Recursive Data Format

- Mapping 2-D (Matrix) to 1-D (RAM)
  - C/Fortran do not map well
- Space-Filling Curve Approximation
  - Recursive **Tiling**
- Enables
  - Streaming/pre-fetching
  - **Dual core** “scaling”



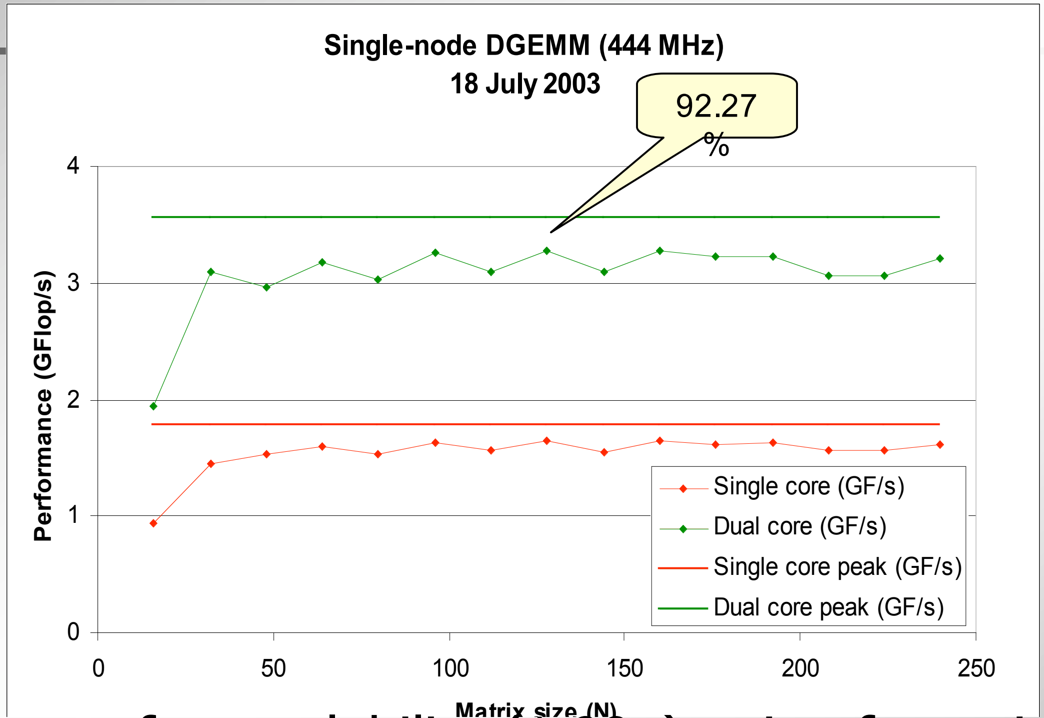
Dual Core

- Why?
- It's a effortless way to double your performance

Dual Core

- Why?
- It exploits the architecture and may allow one to double the performance of their code in some cases/regions

Single-Node DGEMM Performance at 92% of Peak



- Near-perfect scalability (1.99×) going from single-core to dual-core
- Dual-core code delivers 92.27% of peak flops (8 flop/pclk)
- Performance (as fraction of peak) competitive with that of Power3 and Power4

Points to consider

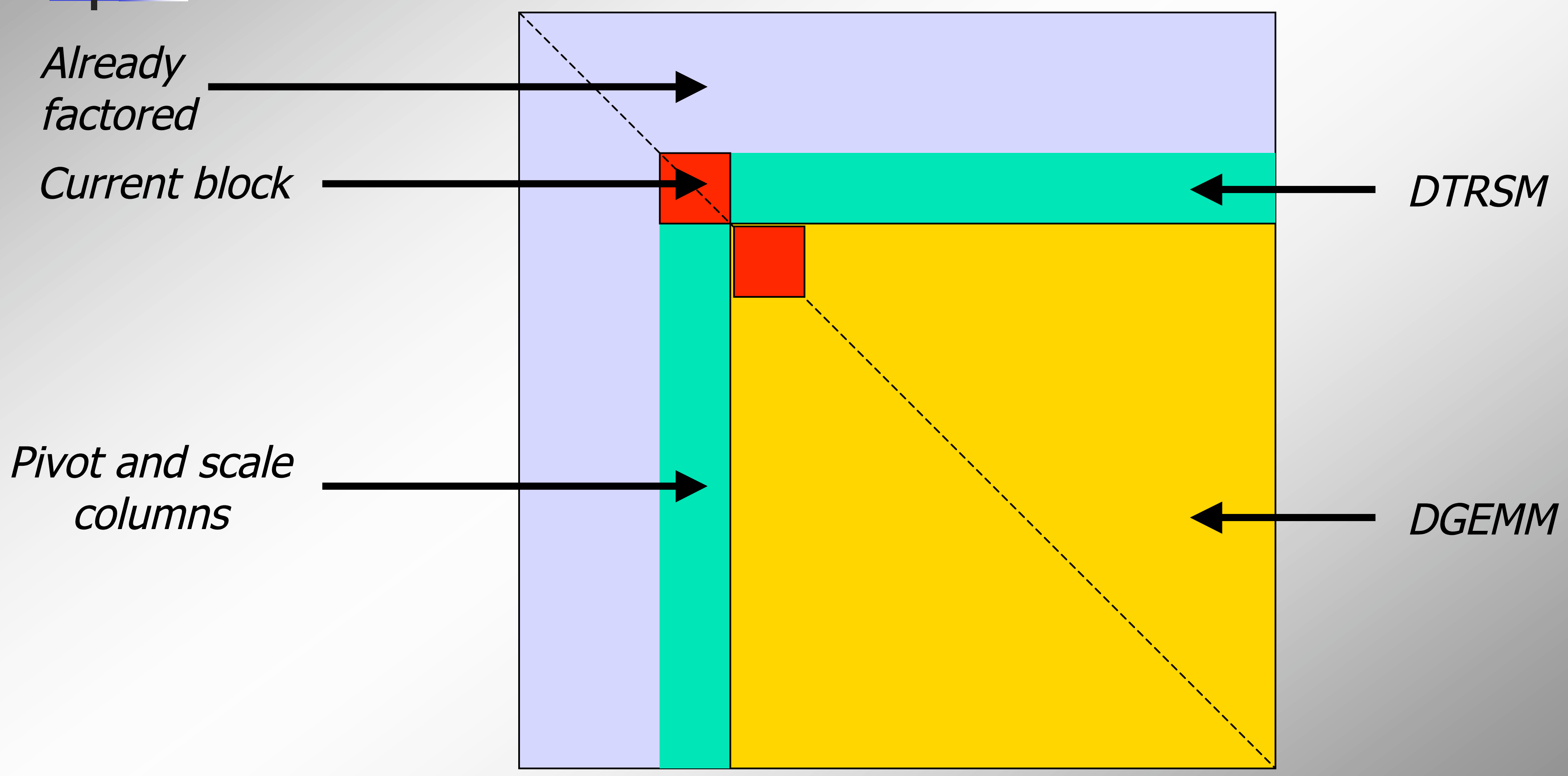
- Code fusion can enable one to
  - Perform a data re-format and/or make effective use of both cores for an operation
- The architecture is very rich
  - Corner cases have to be handled
  - Can be very powerful
  - Helpful in understanding performance
  - Semi-esoteric improvements exist
    - Fine-grained L1 data cache control

What More Could We Want?

- Open up the cache architecture more
  - It would be good if the library writer could specify that a particular access would be a miss in L1, or a hit in L3, for example
  - Expose more microarchitectural constraints to the compiler
  - Example: maximum number of L1 cache misses before stall
- Better register scheduling algorithms
  - Currently, we have observed excessive spills when using close to all 32 registers

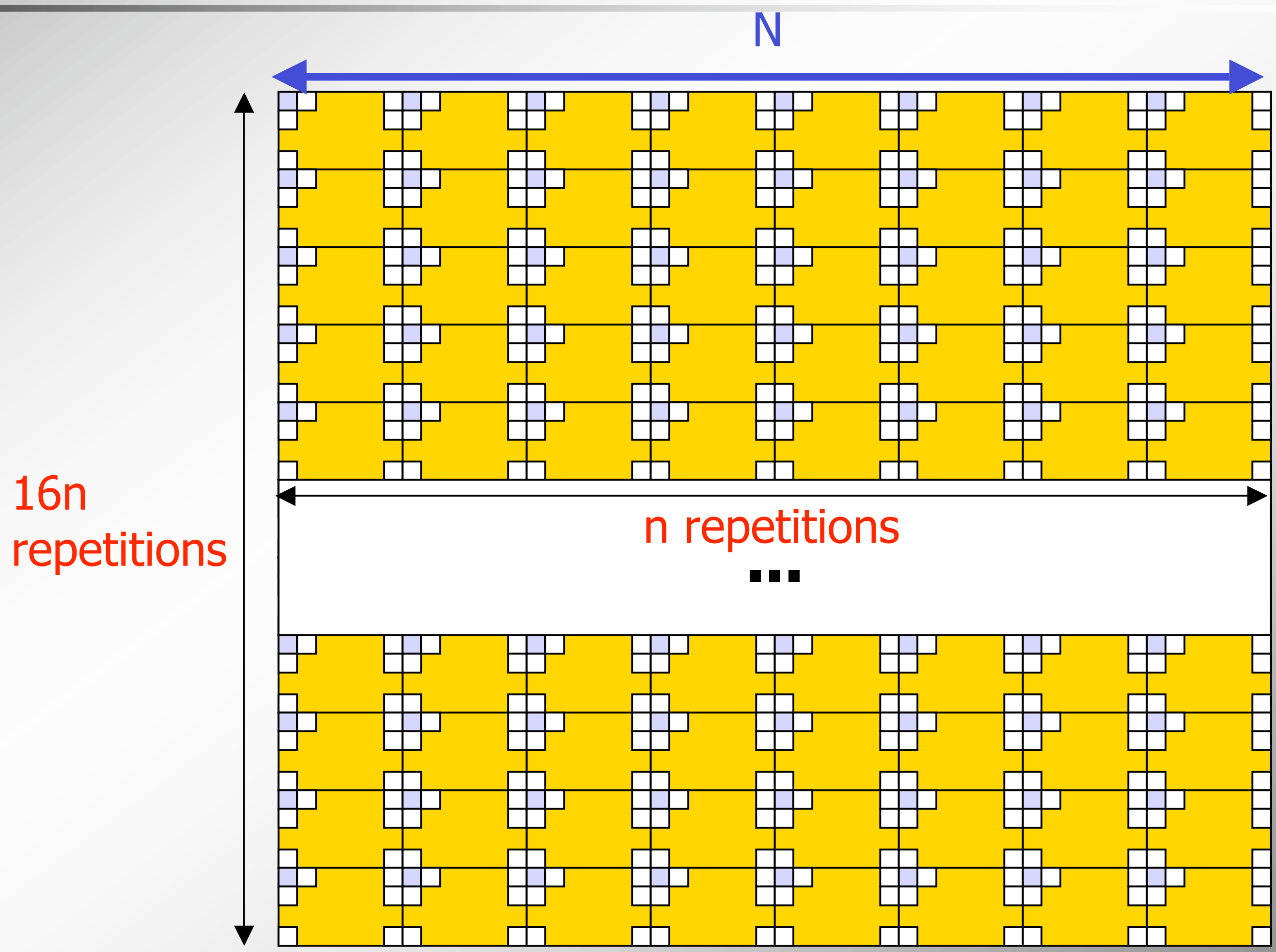
The Linpack Benchmark

LU Factorization: Brief Review



LINPACK

Problem Mapping



Panel Factorization: Option #1

- **Stagger the computations**
- PF Distributed over relatively few processors
- May take as long as several DGEMM updates
- DGEMM load imbalance
  - Block size trades balance for speed
- Use collective communication primitives
  - May require no “holes” in communication fabric

Speed-up Option #2

- **Change the data distribution**
  - Decrease the critical path length
  - Consider the communication abilities of machine
- Complements Option #1
- Memory size (small favors #2; large #1)
  - Memory hierarchy (higher latency: #1)
- The two options can be used in concert

Communication Routines

- Broadcasts precede DGEMM update
- Needs to be architecturally aware
  - Multiple “pipes” connect processors
- Physical to logical mapping
- Careful orchestration is required to take advantage of machines considerable abilities
- See: MPI Presentation (MPI_Bcast)

What Else?

- **It’s a(n) …**
  - FPU Test
  - Memory Test
  - Power Test
  - Torus Test
  - Mode Test (Virtual/Co-)

Conclusion: Scaling

Conclusion: Scaling

Conclusion: Scaling

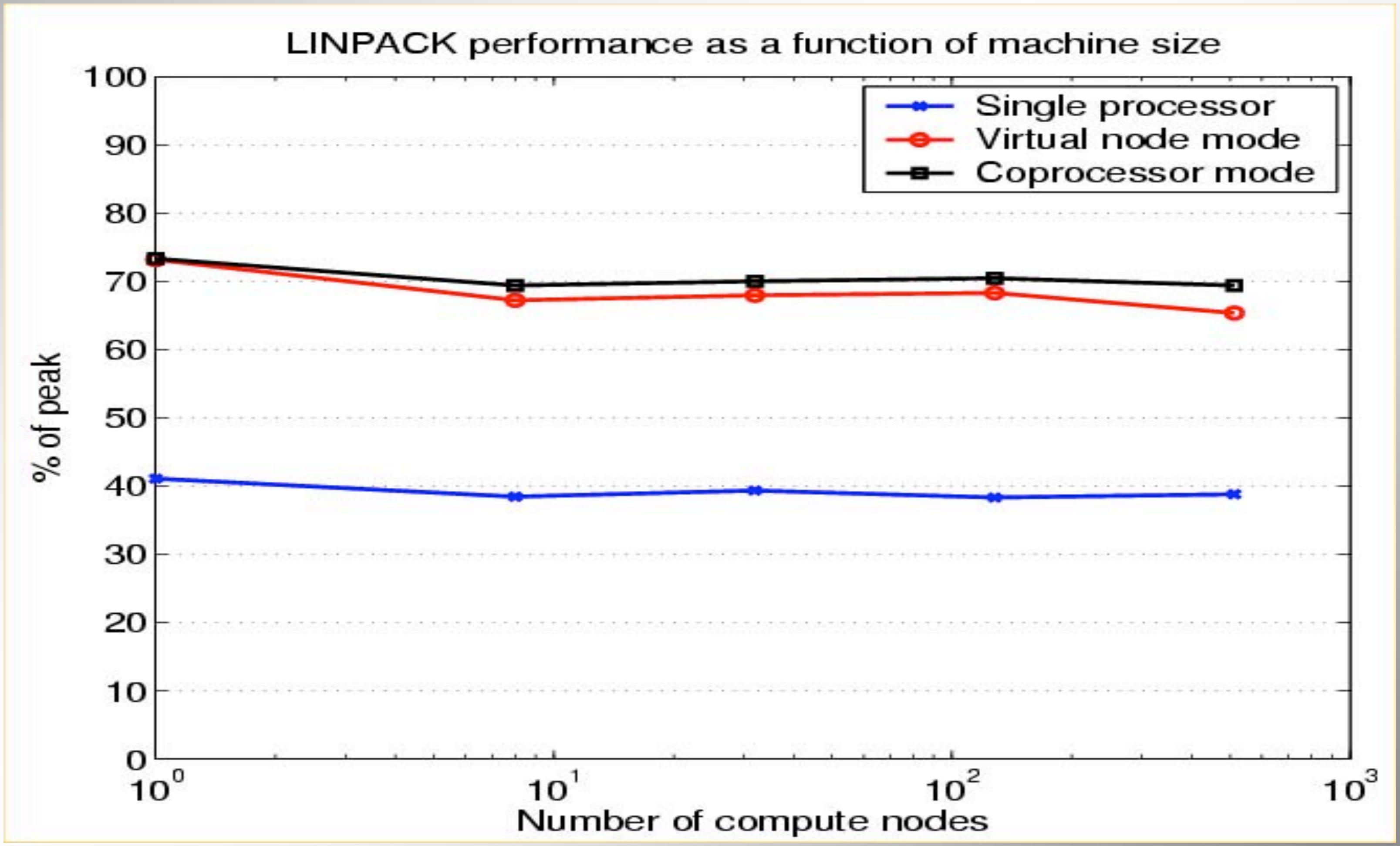
- Contributions to lack of “flat” scaling
  - Time spent tuning for a particular configuration
  - Different driver versions evidence different characteristics
    - Runs performed at different stages
  - Physical layout of machine
    - Aspect ratio

Conclusion

- #73 in TOP500 List (11/2003)
  - Limited Machine Access Time
    - Made analysis/model more important
- #4 (4096 DD1) & #8 (2048 DD2) on 6/2004 TOP500
- **#1 on 11/2004 TOP500**
  - Also:#8 (4096 DD1) & #15 (2048 DD2)

Conclusion: **Breakdown** (old data)

What about VNM?



Additional Conclusions

- Models, extrapolated data
  - Use models to the extent that the architecture and algorithm are understood
  - Extrapolate from small processor sets
  - Vary as many (yes) parameters as possible at the same time
    - Consider how they interact and how they don't
    - Also remember that instruments affect timing
      - Often can compensate (incorrect answer results)
  - Utilize observed "eccentricities" with caution (MPI_Reduce)

Current Fronts

- HPC Challenge Benchmark Suite
  - STREAMS, HPL, etc.
- HPCS Productivity Benchmarks
- Math Libraries
- Focused Feedback to Toronto
- PERCS Compiler/Persistent Optimization
- Linpack Algorithm on Other Machines

Benchmarks on BG/L: Parallel and Serial

John A. Gunnels
Mathematical Sciences Dept.
IBM T. J. Watson Research Center

TAU Performance Tools

Sameer Shende, Allen D. Malony, and Alan Morris

{sameer, malony, amorris}@cs.uoregon.edu

Department of Computer and Information Science

NeuroInformatics Center

University of Oregon

UNIVERSITY
OF OREGON

Acknowledgements

- Pete Beckman, ANL
- Suravee Suthikulpanit, U. Oregon
- Aroon Nataraj, U. Oregon
- Katherine Riley, ANL

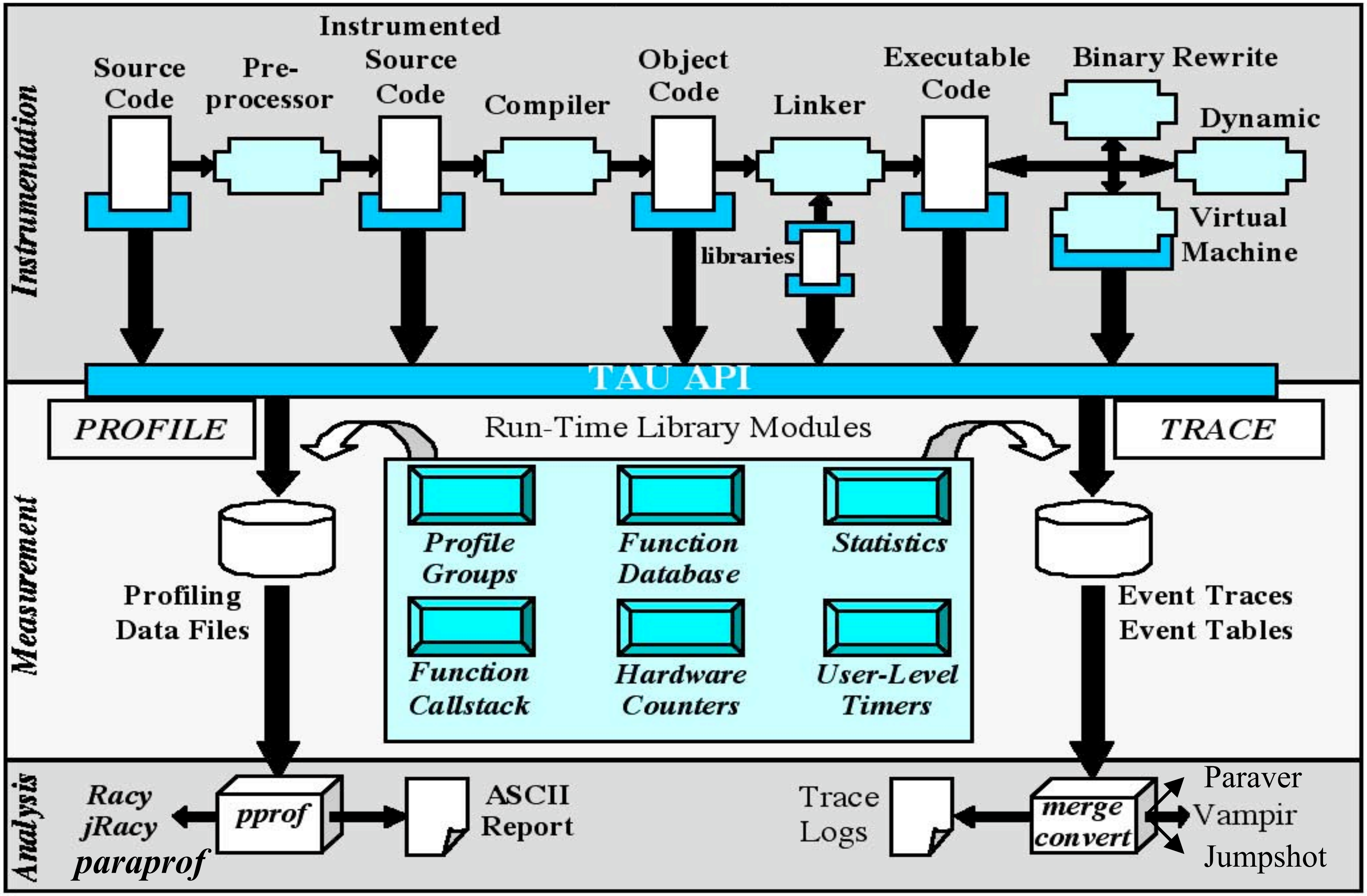
Outline

- Overview of features
- Instrumentation
- Measurement
- Analysis tools
- Linux kernel profiling with TAU

TAU Performance System Framework

- Tuning and Analysis Utilities
- Performance system framework for scalable parallel and distributed high-performance computing
- Targets a general complex system computation model
  - nodes / contexts / threads
  - Multi-level: system / software / parallelism
  - Measurement and analysis abstraction
- Integrated toolkit for performance instrumentation, measurement, analysis, and visualization
  - Portable, configurable performance profiling/tracing facility
  - Open software approach
- University of Oregon, LANL, FZJ Germany
- http://www.cs.uoregon.edu/research/paracomp/tau

TAU Performance System Architecture



TAU Instrumentation Approach

- Support for standard program events
  - Routines
  - Classes and templates
  - Statement-level blocks
- Support for user-defined events
  - Begin/End events (“user-defined timers”)
  - Atomic events (e.g., size of memory allocated/freed)
  - Selection of event statistics
- Support definition of “semantic” entities for mapping
- Support for event groups
- Instrumentation optimization (eliminate instrumentation in lightweight routines)

TAU Instrumentation

- Flexible instrumentation mechanisms at multiple levels
  - Source code
    - manual (TAU API, TAU Component API)
    - automatic
      - C, C++, F77/90/95 (Program Database Toolkit (*PDT*))
      - OpenMP (directive rewriting (*Opari*), *POMP spec*)
  - Object code
    - pre-instrumented libraries (e.g., MPI using *PMPI*)
    - statically-linked and dynamically-linked
  - Executable code
    - dynamic instrumentation (pre-execution) (*DynInstAPI*)
    - virtual machine instrumentation (e.g., Java using *JVMPI*)
  - Proxy Components

Using TAU – A tutorial

- Configuration
- Instrumentation
  - Manual
  - MPI – Wrapper interposition library
  - PDT- Source rewriting for C,C++, F77/90/95
  - OpenMP – Directive rewriting
  - Component based instrumentation – Proxy components
  - Binary Instrumentation
    - DyninstAPI – Runtime instrumentation/Rewriting binary
    - Java – Runtime instrumentation
    - Python – Runtime instrumentation
- Measurement
- Performance Analysis

TAU Measurement System Configuration

- configure [OPTIONS]

{-c++=<CC>, -cc=<cc>}	Specify C++ and C compilers
{-pthread, -sproc}	Use pthread or SGI sproc threads
-openmp	Use OpenMP threads
-jdk=<dir>	Specify Java instrumentation (JDK)
-opari=<dir>	Specify location of Opari OpenMP tool
-papi=<dir>	Specify location of PAPI
-pdt=<dir>	Specify location of PDT
-dyninst=<dir>	Specify location of DynInst Package
-mpi[inc/lib]=<dir>	Specify MPI library instrumentation
-shmem[inc/lib]=<dir>	Specify PSHMEM library instrumentation
-python[inc/lib]=<dir>	Specify Python instrumentation
-epilog=<dir>	Specify location of EPILOG
-vtf=<dir>	Specify location of VTF3 trace package
-arch=<architecture> (bgl,ibm64,ibm64linux…)	Specify architecture explicitly

TAU Measurement System Configuration

- configure [OPTIONS]
  - -TRACE — Generate binary TAU traces
  - -PROFILE (default) — Generate profiles (summary)
  - -PROFILECALLPATH — Generate call path profiles
  - -PROFILEPHASE — Generate phase based profiles
  - -PROFILEMEMORY — Track heap memory for each routine
  - -MULTIPLECOUNTERS — Use hardware counters + time
  - -COMPENSATE — Compensate timer overhead
  - -CPUTIME — Use usertime+system time
  - -PAPIWALLCLOCK — Use PAPI's wallclock time
  - -PAPIVIRTUAL — Use PAPI's process virtual time
  - -SGITIMERS — Use fast IRIX timers
  - -LINUXTIMERS — Use fast x86 Linux timers

TAU Measurement Configuration – Examples

- ./configure –arch=bgl –mpi –pdt=/usr/pdtoolkit-3.3.1 -pdt_c++=xlC
  - Use IBM BlueGene/L arch, XL compilers, MPI and PDT
  - Builds <tau>/bgl/bin/tau_instrumentor (executes on the front-end) and <tau>/bgl/lib/Makefile.tau-mpi-pdt stub
- ./configure –TRACE –PROFILE –arch=bgl –mpi
  - Enable both TAU profiling and tracing
- ./configure -c++=xlC_r -cc=xlc_r -mpi –pdt=/home/pdtoolkit-3.3.1 –TRACE –vtf=/usr/vtf3-1.33
  - Use IBM's xlC_r and xlc_r compilers with VTF3, PDT, MPI packages and multiple counters for measurements on the ppc64 front-end node
- Typically configure multiple measurement libraries

TAU Performance Framework Interfaces

- PDT [U. Oregon, LANL, FZJ] for instrumentation of C++, C99, F95 source code
- PAPI [UTK] & PCL[FZJ] for accessing hardware performance counters data
- DyninstAPI [U. Maryland, U. Wisconsin] for runtime instrumentation
- KOJAK [FZJ, UTK]
  - Epilog trace generation library
  - CUBE callgraph visualizer
  - Opari OpenMP directive rewriting tool
- Vampir/Intel® Trace Analyzer [Pallas/Intel]
- VTF3 trace generation library for Vampir [TU Dresden] (available from TAU website)
- Paraver trace visualizer [CEPBA]
- Jumpshot-4 trace visualizer [MPICH, ANL]
- JVMPI from JDK for Java program instrumentation [Sun]
- Paraprof profile browser/PerfDMF database supports:
  - TAU format
  - Gprof [GNU]
  - HPM Toolkit [IBM]
  - MpiP [ORNL, LLNL]
  - Dynaprof [UTK]
  - PSRun [NCSA]
- PerfDMF database can use Oracle, MySQL or PostgreSQL (IBM DB2 support planned)

Memory Profiling in TAU

- Configuration option –PROFILEMEMORY
  - Records global heap memory utilization for each function
  - Takes one sample at beginning of each function and associates the sample with function name
  - Independent of instrumentation/measurement options selected
  - No need to insert macros/calls in the source code
  - User defined atomic events appear in profiles/traces

Memory Profiling in TAU

```
Sorted By: number of userEvents
   ---------------------------------------------------------------------------------------------

   NumSamples        Max              Min              Mean             Std. Dev         Name

   ---------------------------------------------------------------------------------------------
   252032            2022.7           1181.2           1534.3           410.04           MODULEHYDRO_1D::HYDRO_1D   - Heap Memory (KB)
   252032            2022.8           1181.7           1534.3           410.04           MODULEINTRFC::INTRFC   - Heap Memory (KB)
   104559            2023.2           331.13           1526.6           409.54           MODULEEOS3D::EOS3D   - Heap Memory (KB)
   63008             2022.7             1182           1534.3           410.01           MODULEUPDATE_SOLN::UPDATE_SOLN   - Heap Memory (KB)
   55545             2023.3           333.07           1514.2           408.31           DBASETREE::DBASENEIGHBORBLOCKLIST   - Heap Memory (KB)
   51374               2023           1179.4           1497.7           402.53           AMR_PROLONG_GEN_UNK_FUN   - Heap Memory (KB)
   42120             2022.7           1187.5           1533.5           409.83           ABUNDANCE_RESTRICT   - Heap Memory (KB)
   41958               2023           346.12           1514.9           408.39           AMR_RESTRICT_UNK_FUN   - Heap Memory (KB)
   31832             2022.8           1187.4           1534.1           409.91           AMR_RESTRICT_RED   - Heap Memory (KB)
   31504             2022.7           1181.8           1534.3           410.04           DIFFUSE   - Heap Memory (KB)
   26042               2023           1179.2           1501.9           403.61           AMR_PROLONG_UNK_FUN   - Heap Memory (KB)
```

Flash2 code profile on IBM BlueGene/L [MPI rank 0]

Memory Profiling in TAU

- Instrumentation based observation of global heap memory (not per function)
  - call TAU_TRACK_MEMORY()
    - Triggers one sample every 10 secs
  - call TAU_TRACK_MEMORY_HERE()
    - Triggers sample at a specific location in source code
  - call TAU_SET_INTERRUPT_INTERVAL(seconds)
    - To set inter-interrupt interval for sampling
  - call TAU_DISABLE_TRACKING_MEMORY()
    - To turn off recording memory utilization
  - call TAU_ENABLE_TRACKING_MEMORY()
    - To re-enable tracking memory utilization

Profile Measurement – Three Flavors

- Flat profiles
  - Time (or counts) spent in each routine (nodes in callgraph).
  - Exclusive/inclusive time, no. of calls, child calls
  - E.g,: MPI_Send, foo, …
- Callpath Profiles
  - Flat profiles, **plus**
  - Sequence of actions that led to poor performance
  - Time spent along a calling path (edges in callgraph)
  - E.g., “main=> f1 => f2 => MPI_Send” shows the time spent in MPI_Send when called by f2, when f2 is called by f1, when it is called by main. Depth of this callpath = 4 (TAU_CALLPATH_DEPTH environment variable)
- Phase based profiles
  - Flat profiles, **plus**
  - Flat profiles under a phase (nested phases are allowed)
  - Default “main” phase has all phases and routines invoked outside phases
  - Supports static or dynamic (per-iteration) phases
  - E.g., “IO => MPI_Send” is time spent in MPI_Send in IO phase

TAU Timers and Phases

- Static timer
  - Shows time spent in all invocations of a routine (foo)
  - E.g., “foo()” 100 secs, 100 calls
- Dynamic timer
  - Shows time spent in each invocation of a routine
  - E.g., “foo() 3” 4.5 secs, “foo 10” 2 secs (invocations 3 and 10 respectively)
- Static phase
  - Shows time spent in all routines called (directly/indirectly) by a given routine (foo)
  - E.g., “foo() => MPI_Send()” 100 secs, 10 calls shows that a total of 100 secs were spent in MPI_Send() when it was called by foo.
- Dynamic phase
  - Shows time spent in all routines called by a given invocation of a routine.
  - E.g., “foo() 4 => MPI_Send()” 12 secs, shows that 12 secs were spent in MPI_Send when it was called by the 4th invocation of foo.

Flat Profile – Pprof Profile Browser

- Intel Linux cluster
- F90 + MPICH
- Profile
  - Node
  - Context
  - Thread
- Events
  - code
  - MPI

emacs@neutron.cs.uoregon.edu

Buffers Files Tools Edit Search Mule Help

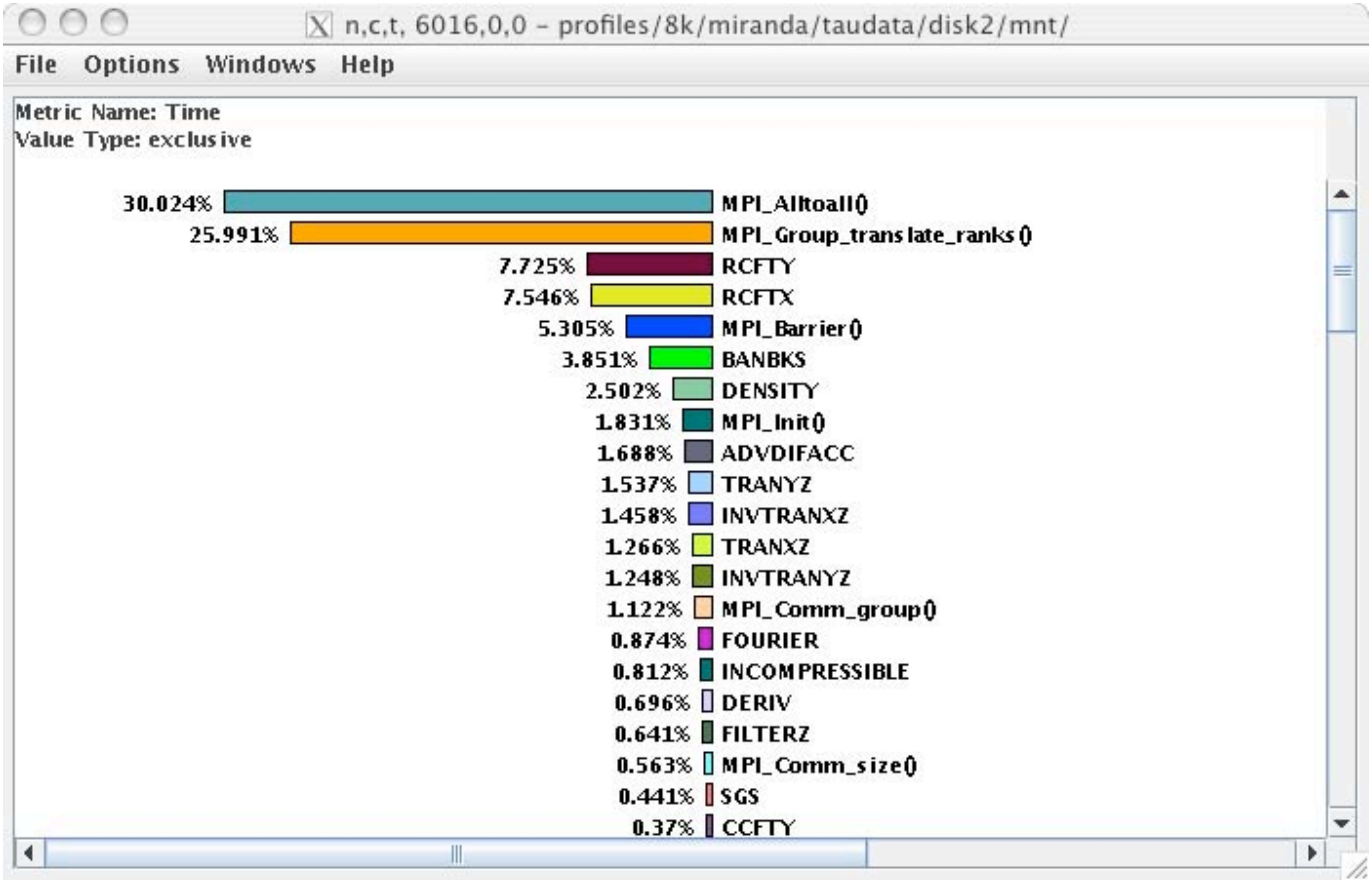
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:

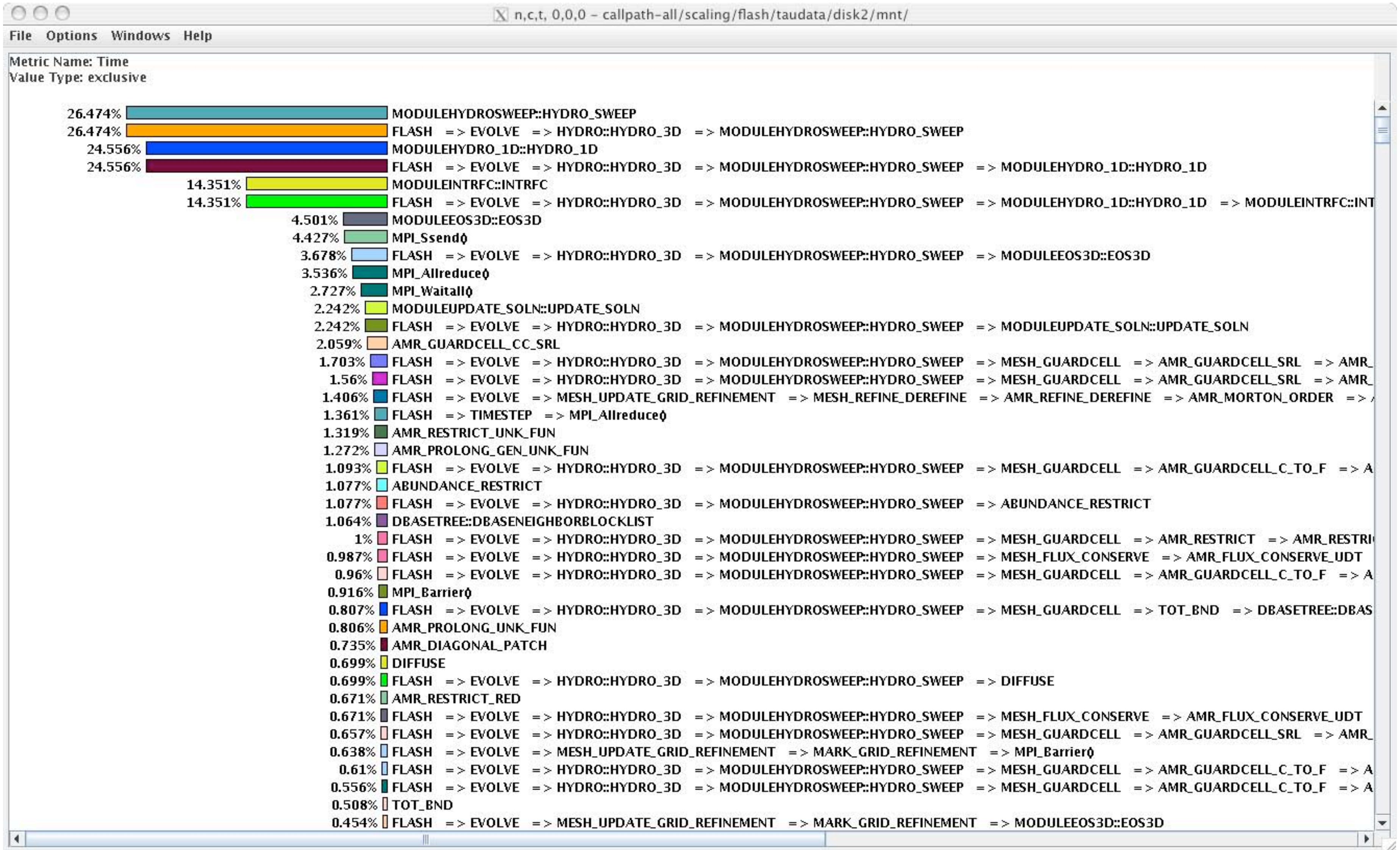
%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive usec/call	Name
100.0	1	3:11.293	1	15	191293269	applu
99.6	3,667	3:10.463	3	37517	63487925	bcast_inputs
67.1	491	2:08.326	37200	37200	3450	exchange_1
44.5	6,461	1:25.159	9300	18600	9157	buts
41.0	1:18.436	1:18.436	18600	0	4217	MPI_Recv()
29.5	6,778	56,407	9300	18600	6065	blts
26.2	50,142	50,142	19204	0	2611	MPI_Send()
16.2	24,451	31,031	301	602	103096	rhs
3.9	7,501	7,501	9300	0	807	jacld
3.4	838	6,594	604	1812	10918	exchange_3
3.4	6,590	6,590	9300	0	709	jacu
2.6	4,989	4,989	608	0	8206	MPI_Wait()
0.2	0.44	400	1	4	400081	init_comm
0.2	398	399	1	39	399634	MPI_Init()
0.1	140	247	1	47616	247086	setiv
0.1	131	131	57252	0	2	exact
0.1	89	103	1	2	103168	erhs
0.1	0.966	96	1	2	96458	read_input
0.0	95	95	9	0	10603	MPI_Bcast()
0.0	26	44	1	7937	44878	error
0.0	24	24	608	0	40	MPI_Irecv()
0.0	15	15	1	5	15630	MPI_Finalize()
0.0	4	12	1	1700	12335	setbv
0.0	7	8	3	3	2893	l2norm
0.0	3	3	8	0	491	MPI_Allreduce()
0.0	1	3	1	6	3874	pintgr
0.0	1	1	1	0	1007	MPI_Barrier()
0.0	0.116	0.837	1	4	837	exchange_4
0.0	0.512	0.512	1	0	512	MPI_Keyval_create()
0.0	0.121	0.353	1	2	353	exchange_5
0.0	0.024	0.191	1	2	191	exchange_6
0.0	0.103	0.103	6	0	17	MPI_Type_contiguous()

--:-- NPB_LU.out (Fundamental)--L8--Top--

Flat Profile – TAU's Paraprof Profile Browser



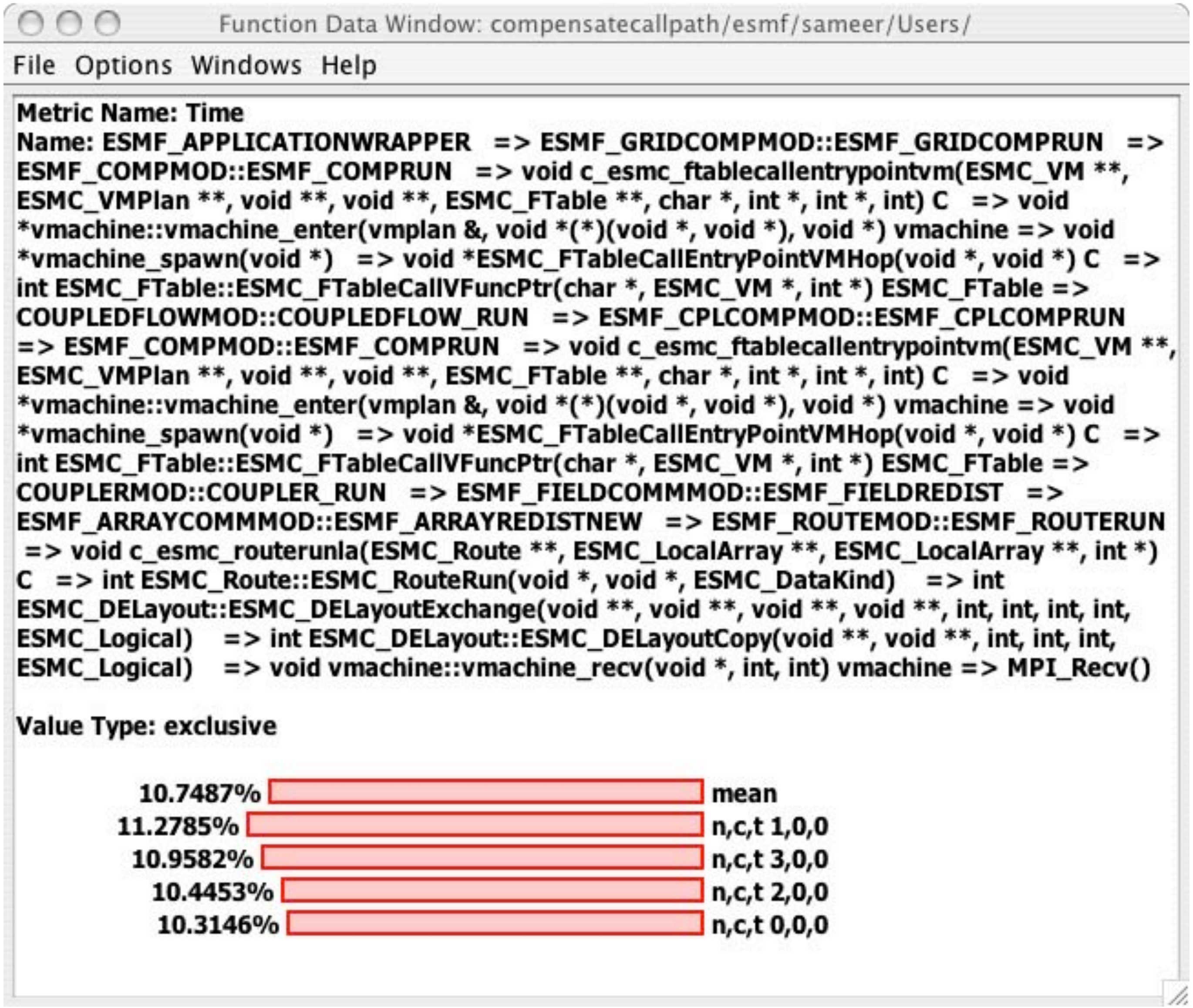
Callpath Profile



Callpath Profile - parent/node/child view

```
Metric Name: Time
Sorted By: exclusive
Units: seconds

     Exclusive     Inclusive     Calls/Tot.Calls                Name[id]
     -------------------------------------------------------------------------------

     1.8584        1.8584        1196/13188                     TOKEN_MODULE::TOKEN_GS_I  [521]
     0.584         0.584         234/13188                      TOKEN_MODULE::TOKEN_GS_L  [544]
     25.0819       25.0819       11758/13188                    TOKEN_MODULE::TOKEN_GS_R8  [734]
 --> 27.5242       27.5242       13188                          MPI_Waitall()  [525]


     17.9579       39.1657       156/156                        DERIVATIVE_MODULE::DERIVATIVES_NOFACE  [841]
 --> 17.9579       39.1657       156                            DERIVATIVE_MODULE::DERIVATIVES_FACE  [843]
     0.0156        0.0195        312/312                        TIMER_MODULE::TIMERSET  [77]
     0.1133        9.1269        2340/2340                      MESSAGE_MODULE::CLONE_GET_R8  [808]
     0.1602        11.4608       4056/4056                      MESSAGE_MODULE::CLONE_PUT_R8  [850]
     0.0059        0.6006        117/117                        MESSAGE_MODULE::CLONE_PUT_I  [856]


     14.1151       21.6209       5/5                            MATRIX_MODULE::MCGDS  [1443]
 --> 14.1151       21.6209       5                              MATRIX_MODULE::CSR_CG_SOLVER  [1470]
     0.0654        1.2617        1005/1005                      TOKEN_MODULE::TOKEN_GET_R8  [769]
     0.0557        5.2714        1005/1005                      TOKEN_MODULE::TOKEN_REDUCTION_R8_S  [1475]
     0.0703        0.9726        1000/1000                      TOKEN_MODULE::TOKEN_REDUCTION_R8_V  [208]
```

Callpath Profiling



Phase Profile – Dynamic Phases

In 51st iteration, time spent in MPI_Waitall was 85.81 secs

Total time spent in MPI_Waitall was 4137.9 secs across all 92 iterations

X Call Path Data n,c,t, 0,0,0 – phase/mfix/taudata/disk2/mnt/

File Options Windows Help

Metric Name: Time
Sorted By: exclusive
Units: seconds

47.376	47.376	15712.0/1345134.0	ITERATE	47[6980]
65.217	65.217	21232.0/1345134.0	ITERATE	48[7116]
55.321	55.321	17888.0/1345134.0	ITERATE	49[7252]
51.351	51.351	16592.0/1345134.0	ITERATE	50[7388]
85.81	85.81	28208.0/1345134.0	ITERATE	51[7524]
75.069	75.069	24384.0/1345134.0	ITERATE	52[7670]
78.938	78.938	25728.0/1345134.0	ITERATE	53[7806]
69.684	69.684	23104.0/1345134.0	ITERATE	54[7942]
58.461	58.461	19072.0/1345134.0	ITERATE	55[8080]
85.117	85.117	27856.0/1345134.0	ITERATE	56[8216]
47.885	47.885	15504.0/1345134.0	ITERATE	57[8354]
46.436	46.436	14816.0/1345134.0	ITERATE	58[8490]
46.242	46.242	14752.0/1345134.0	ITERATE	59[8636]
45.728	45.728	14640.0/1345134.0	ITERATE	60[8772]
45.244	45.244	14656.0/1345134.0	ITERATE	61[8908]
45.283	45.283	14416.0/1345134.0	ITERATE	62[9044]
61.168	61.168	20032.0/1345134.0	ITERATE	63[9180]
46.992	46.992	15600.0/1345134.0	ITERATE	64[9326]
47.01	47.01	15792.0/1345134.0	ITERATE	65[9462]
44.046	44.046	14608.0/1345134.0	ITERATE	66[9598]
47.424	47.424	15584.0/1345134.0	ITERATE	67[9734]
41.176	41.176	13472.0/1345134.0	ITERATE	68[9870]
51.488	51.488	16880.0/1345134.0	ITERATE	69[10016]
43.714	43.714	14480.0/1345134.0	ITERATE	70[10152]
46.175	46.175	15152.0/1345134.0	ITERATE	71[10288]
45.348	45.348	14864.0/1345134.0	ITERATE	72[10424]
38.728	38.728	12848.0/1345134.0	ITERATE	73[10560]
46	46	15008.0/1345134.0	ITERATE	74[10706]
52.453	52.453	17008.0/1345134.0	ITERATE	75[10842]
44.341	44.341	14496.0/1345134.0	ITERATE	76[10978]
44.288	44.288	14240.0/1345134.0	ITERATE	77[11116]
58.298	58.298	18736.0/1345134.0	ITERATE	78[11252]
48.099	48.099	15584.0/1345134.0	ITERATE	79[11388]
45.351	45.351	14480.0/1345134.0	ITERATE	80[11534]
48.512	48.512	15824.0/1345134.0	ITERATE	81[11670]
41.185	41.185	13408.0/1345134.0	ITERATE	82[11806]
34.789	34.789	11248.0/1345134.0	ITERATE	83[11944]
34.061	34.061	10944.0/1345134.0	ITERATE	84[12080]
33.843	33.843	10960.0/1345134.0	ITERATE	85[12216]
33.182	33.182	10848.0/1345134.0	ITERATE	86[12362]
33.165	33.165	10752.0/1345134.0	ITERATE	87[12498]
29.992	29.992	9632.0/1345134.0	ITERATE	88[12634]
28.337	28.337	9136.0/1345134.0	ITERATE	89[12770]
35.926	35.926	11488.0/1345134.0	ITERATE	90[12906]
36.238	36.238	11648.0/1345134.0	ITERATE	91[13052]
27.385	27.385	8896.0/1345134.0	ITERATE	92[13188]
--> 4137.9	4137.9	1345134.0	MPI_Waitall()[121]	

Using TAU

- Install TAU

  % configure ; make clean install
- Instrument application
  - TAU Profiling API
- Typically modify application makefile
  - include TAU's stub makefile, modify variables
- Set environment variables
  - directory where profiles/traces are to be stored
  - name of merged trace file, retain intermediate trace files, etc.
- Execute application

  % mpirun –np <procs> a.out;
- Analyze performance data
  - paraprof, vampir/traceanalyzer, pprof, paraver …

AutoInstrumentation using TAU_COMPILER

- $(TAU_COMPILER) stub Makefile variable in 2.14+ release
- Invokes PDT parser, TAU instrumentor, compiler through **`tau_compiler.sh`** shell script
- Requires minimal changes to application Makefile
  - Compilation rules are not changed
  - User adds $(TAU_COMPILER) before compiler name
    - F90=mpxlf90
Changes to
F90= $(TAU_COMPILER) mpxlf90
- Passes options from TAU stub Makefile to the four compilation stages
- Uses original compilation command if an error occurs

TAU_COMPILER – Improving Integration in Makefiles

OLD

```
include /usr/tau-
2.14/include/Makefile
CXX = mpCC
F90 = mpxlf90_r
PDTPARSE = $(PDTDIR)/
       $(PDTARCHDIR)/bin/cxxparse
TAUINSTR =
$(TAUROOT)/$(CONFIG_ARCH)/
       bin/tau_instrumentor
CFLAGS = $(TAU_DEFS) $(TAU_INCLUDE)
LIBS = $(TAU_MPI_LIBS) $(TAU_LIBS) -
lm
OBJS = f1.o f2.o f3.o … fn.o

app: $(OBJS)
     $(CXX) $(LDFLAGS) $(OBJS) -o $@
       $(LIBS)
.cpp.o:
     $(PDTPARSE) $<
     $(TAUINSTR) $*.pdb $< -o
       $*.i.cpp -f select.dat
     $(CC) $(CFLAGS) -c $*.i.cpp
```

NEW

```
include /usr/tau-
2.14/include/Makefile
CXX = $(TAU_COMPILER) mpCC
F90 = $(TAU_COMPILER) mpxlf90_r
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

app: $(OBJS)
     $(CXX) $(LDFLAGS) $(OBJS) -o $@
       $(LIBS)
.cpp.o:
     $(CC) $(CFLAGS) -c $<
```

TAU_COMPILER Options

- Optional parameters for $(TAU_COMPILER):
  - -optVerbose Turn on verbose debugging messages
  - -optPdtDir="" PDT architecture directory. Typically $(PDTDIR)/$(PDTARCHDIR)
  - -optPdtF95Opts="" Options for Fortran parser in PDT (f95parse)
  - -optPdtCOpts="" Options for C parser in PDT (cparse). Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
  - -optPdtCxxOpts="" Options for C++ parser in PDT (cxxparse). Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
  - -optPdtF90Parser="" Specify a different Fortran parser. For e.g., f90parse instead of f95parse
  - -optPdtUser="" Optional arguments for parsing source code
  - -optPDBFile="" Specify [merged] PDB file. Skips parsing phase.
  - -optTauInstr="" Specify location of tau_instrumentor. Typically $(TAUROOT)/$(CONFIG_ARCH)/bin/tau_instrumentor
  - -optTauSelectFile="" Specify selective instrumentation file for tau_instrumentor
  - -optTau="" Specify options for tau_instrumentor
  - -optCompile="" Options passed to the compiler. Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
  - -optLinking="" Options passed to the linker. Typically $(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_CXXLIBS)
  - -optNoMpi Removes -l*mpi* libraries during linking (default)
  - -optKeepFiles Does not remove intermediate .pdb and .inst.* files

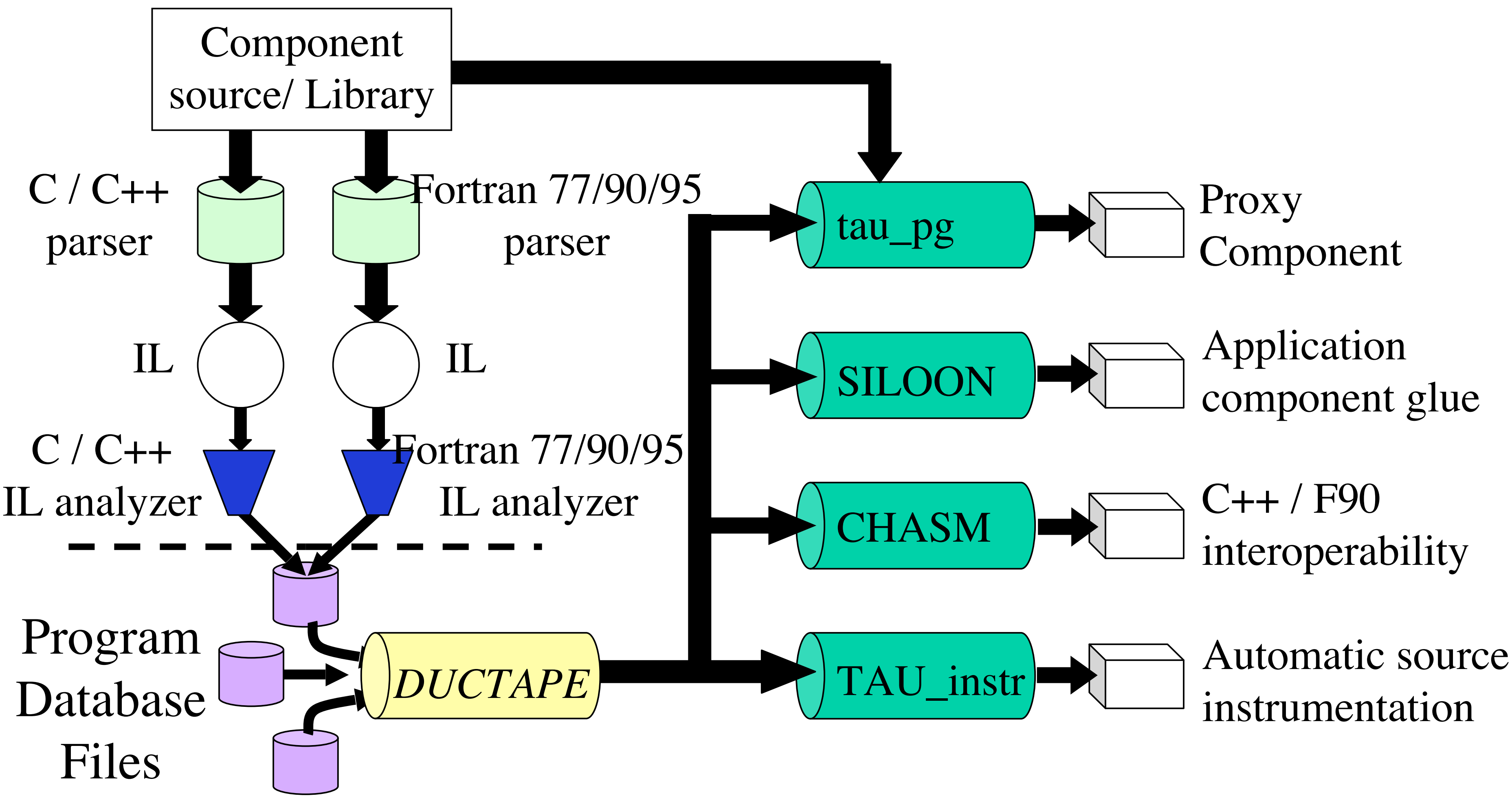
e.g.,

```
OPT=-optTauSelectFile=select.tau -optPDBFile=merged.pdb
F90 = $(TAU_COMPILER) $(OPT) blrts_xlf90
```

Program Database Toolkit (PDT)

- Program code analysis framework
  - develop source-based tools
- *High-level interface* to source code information
- *Integrated toolkit* for source code parsing, database creation, and database query
  - Commercial grade front-end parsers
  - Portable IL analyzer, database format, and access API
  - Open software approach for tool development
- Multiple source languages
- Implement automatic performance instrumentation tools
  - *tau_instrumentor*

Program Database Toolkit



TAU Tracing Enhancements

- Configure TAU with -TRACE –vtf=dir option

  ```
  % configure -TRACE -vtf=<dir> …
  ```

  Generates tau_merge, tau2vtf tools in <tau>/ppc64/bin dir

  ```
  % configure -arch=bgl -TRACE -pdt=<dir>
    -pdt_c++=xlC -mpi
  ```

  Generates library in <tau>/bgl/lib directory

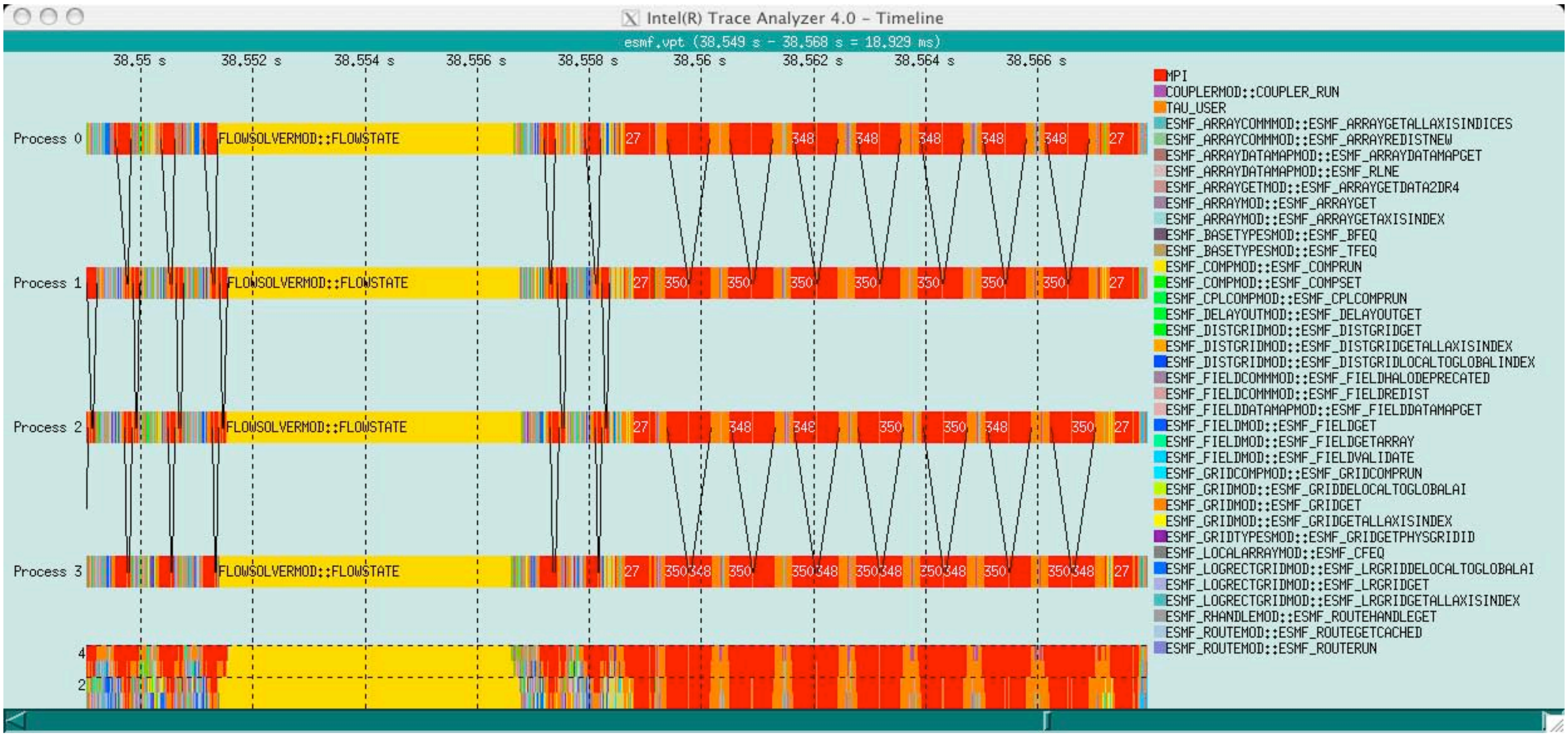
- Execute application

  ```
  % mpirun -partition Pgeneral2 -np 16 -cwd `pwd`
  -exe `pwd`/<app>
  ```

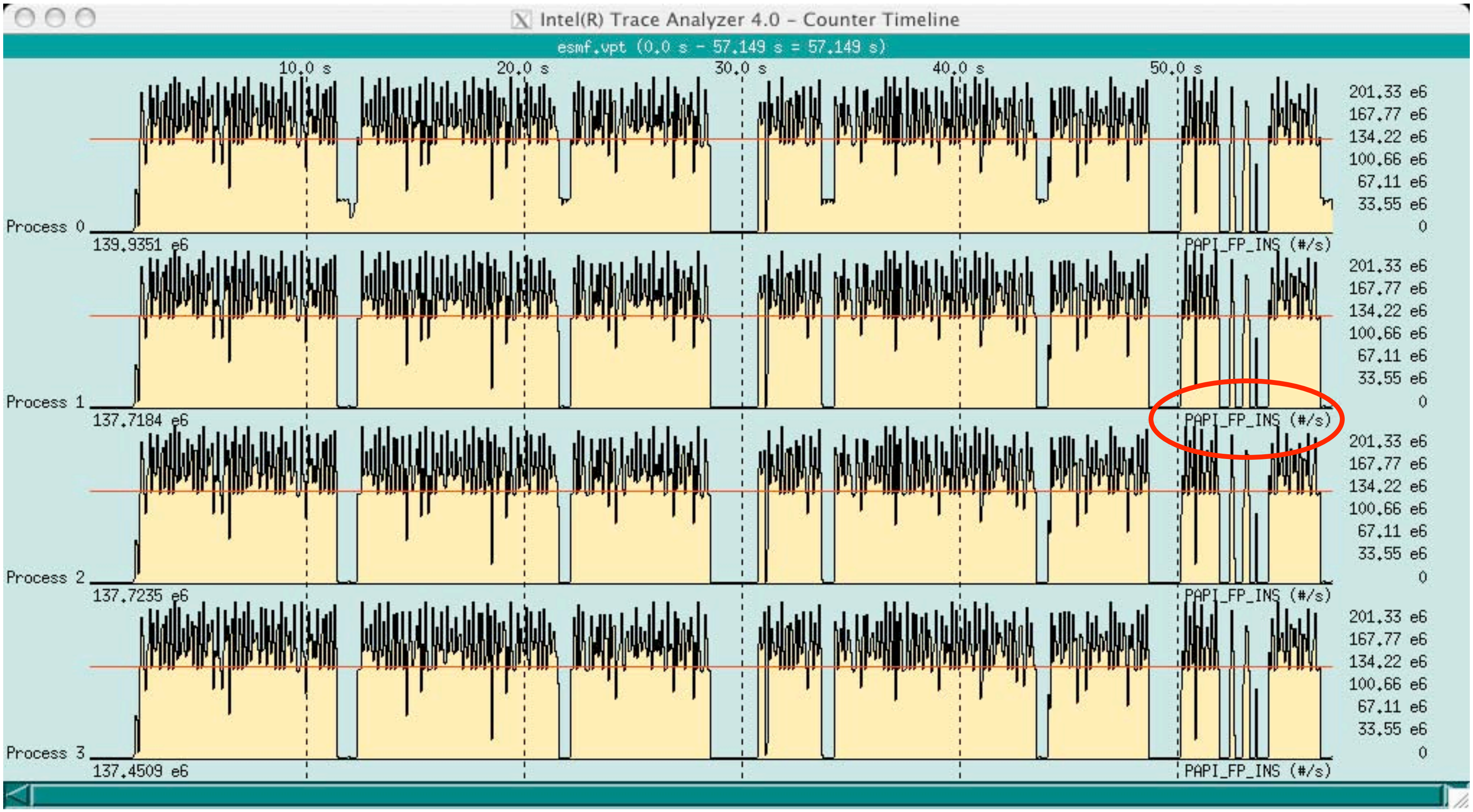
- Merge and convert trace files to VTF3 format

  ```
  % tau_merge *.trc app.trc
  % tau2vtf app.trc tau.edf app.vpt.gz
  % traceanalyzer foo.vpt.gz
  ```

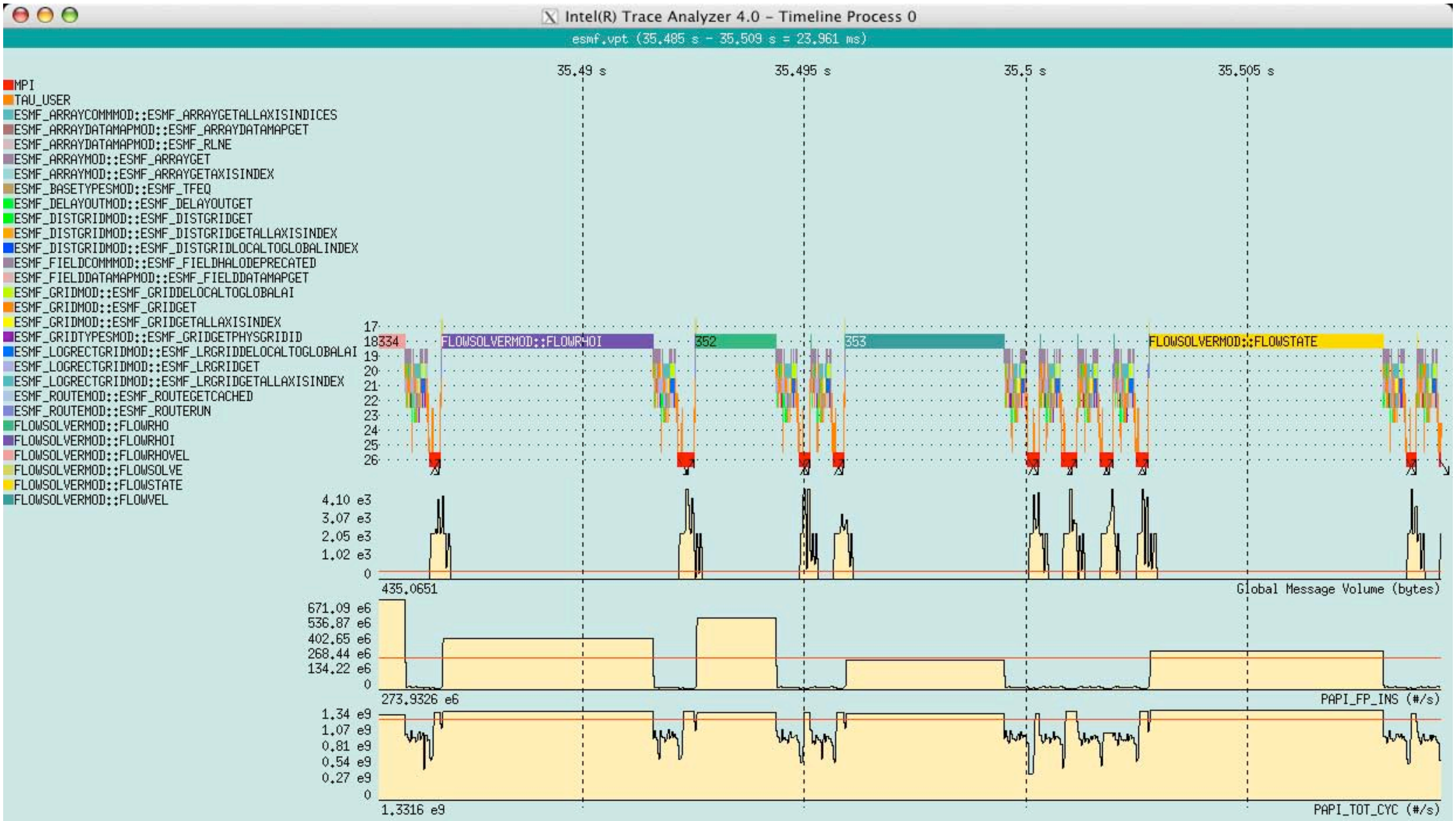
Intel ® Traceanalyzer (Vampir) Global Timeline



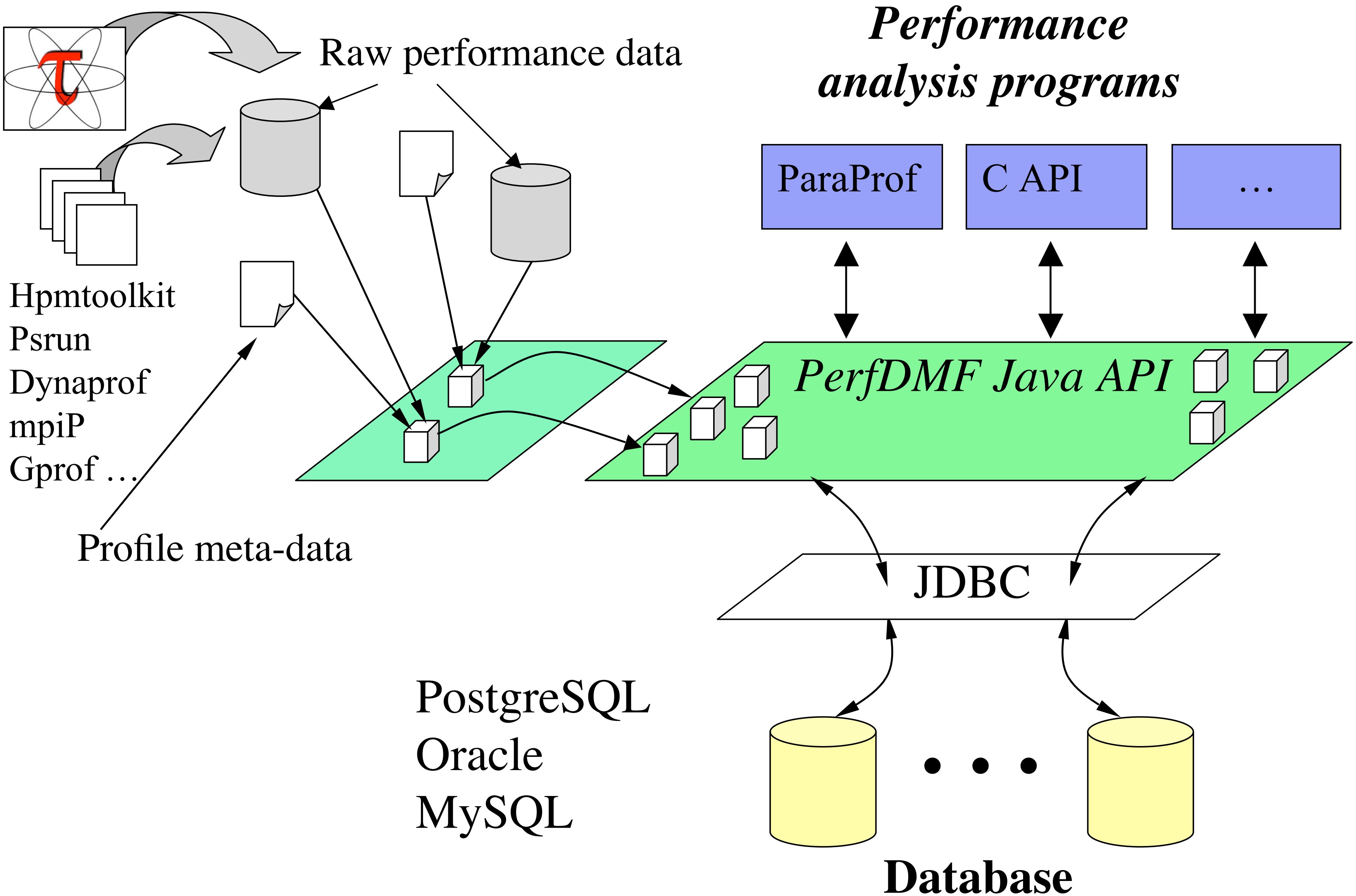
Visualizing TAU Traces with Counters/Samples



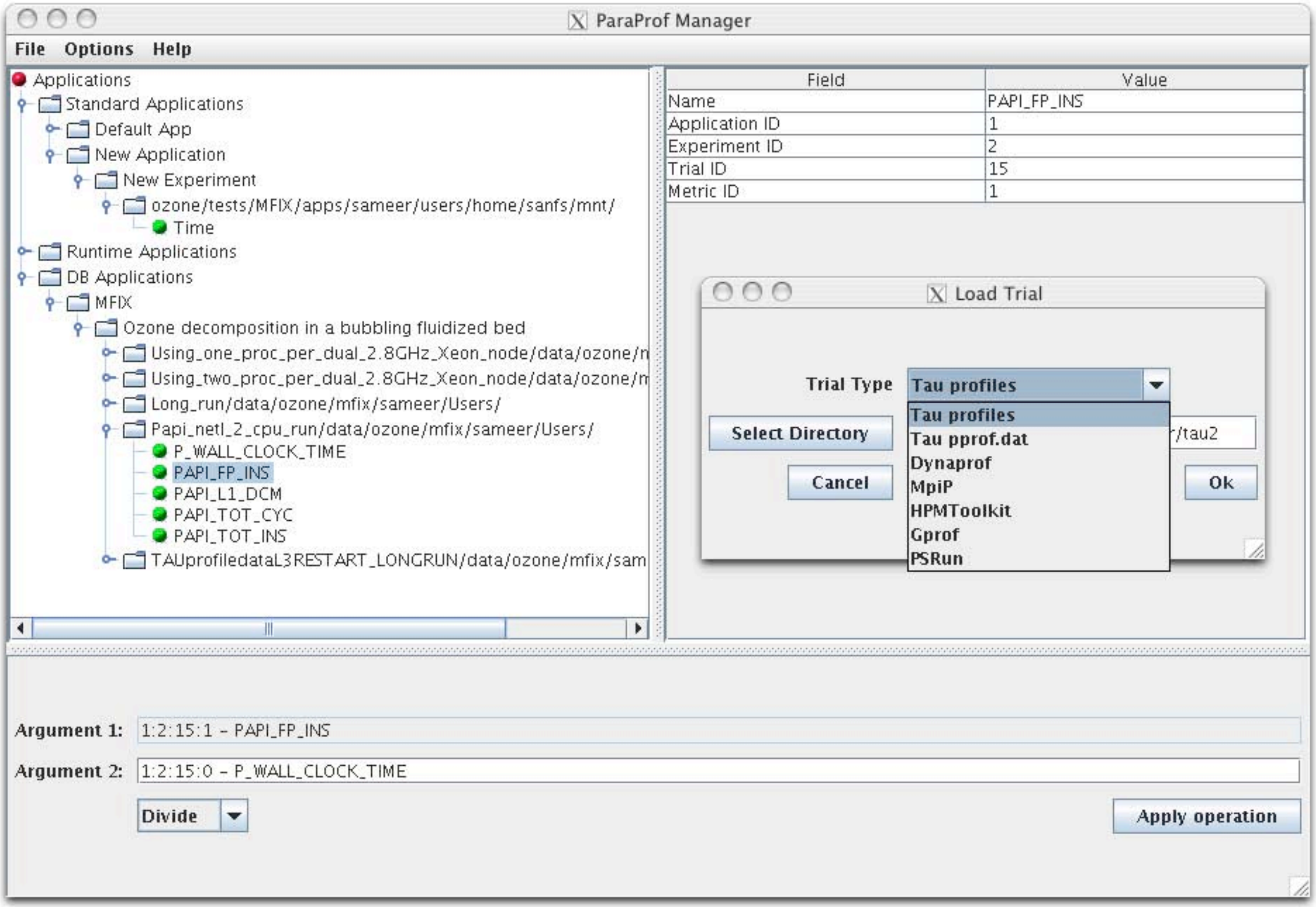
Visualizing TAU Traces with Counters/Samples



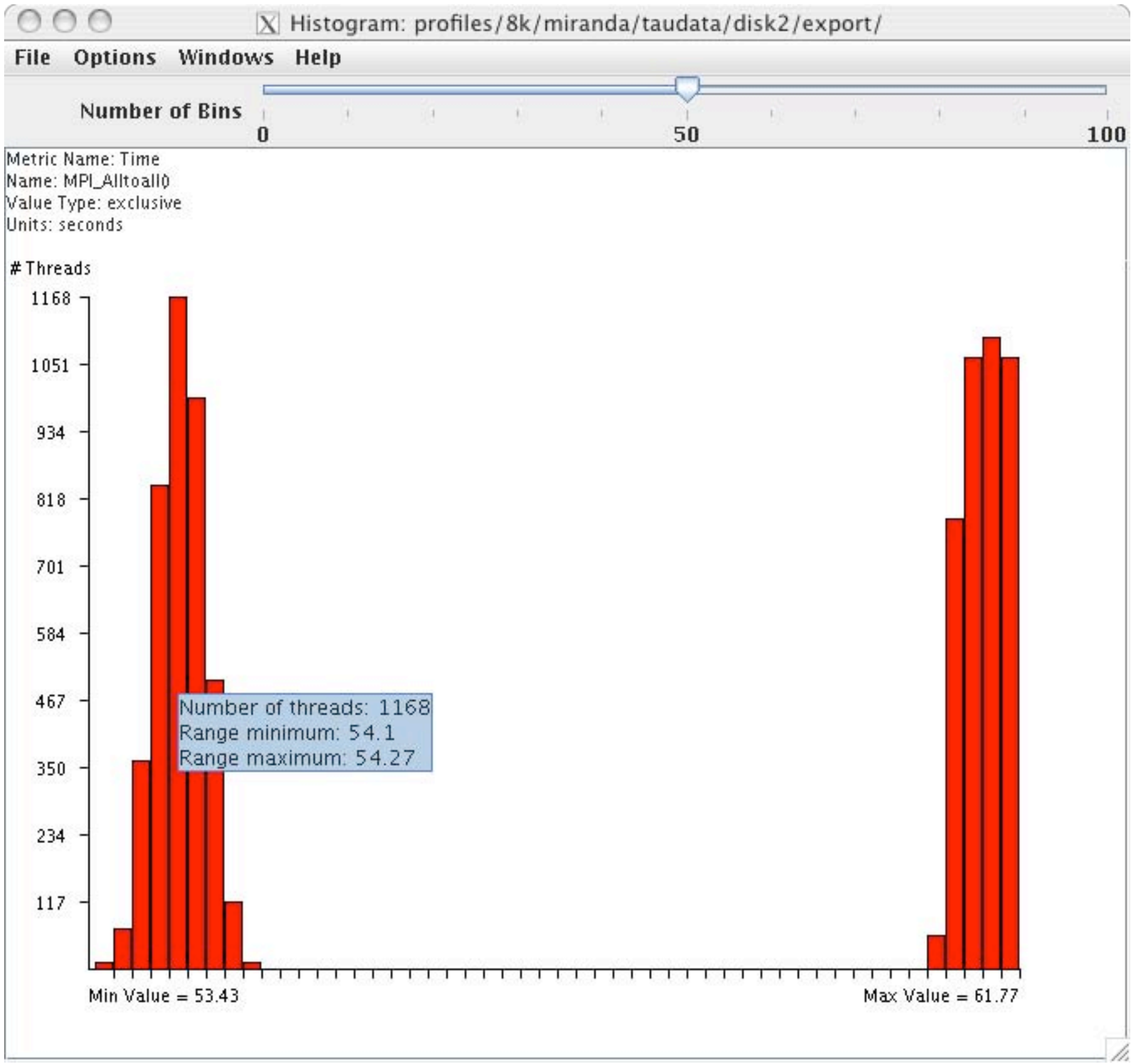
TAU Performance Data Management Framework



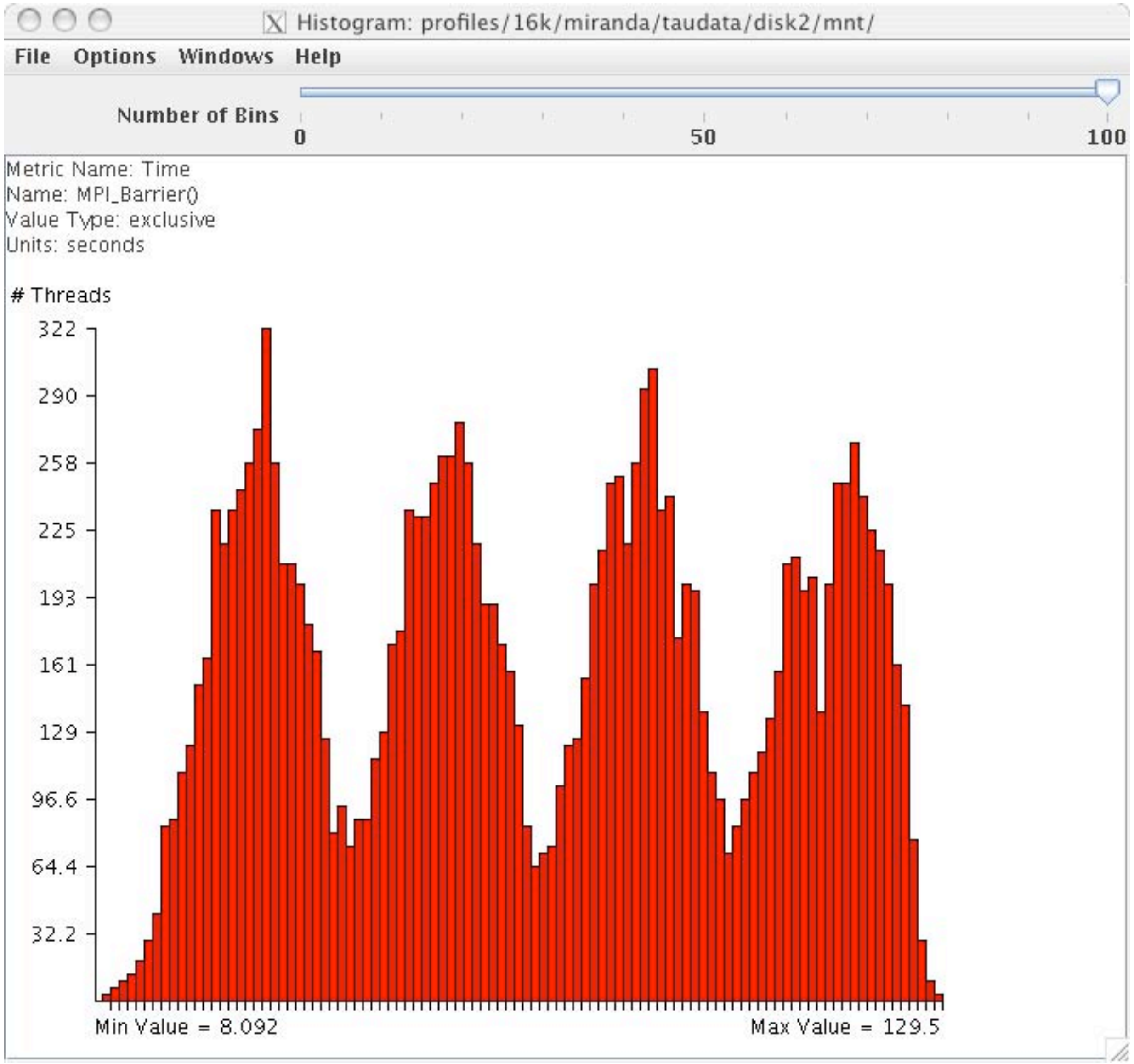
Paraprof Manager – Performance Database



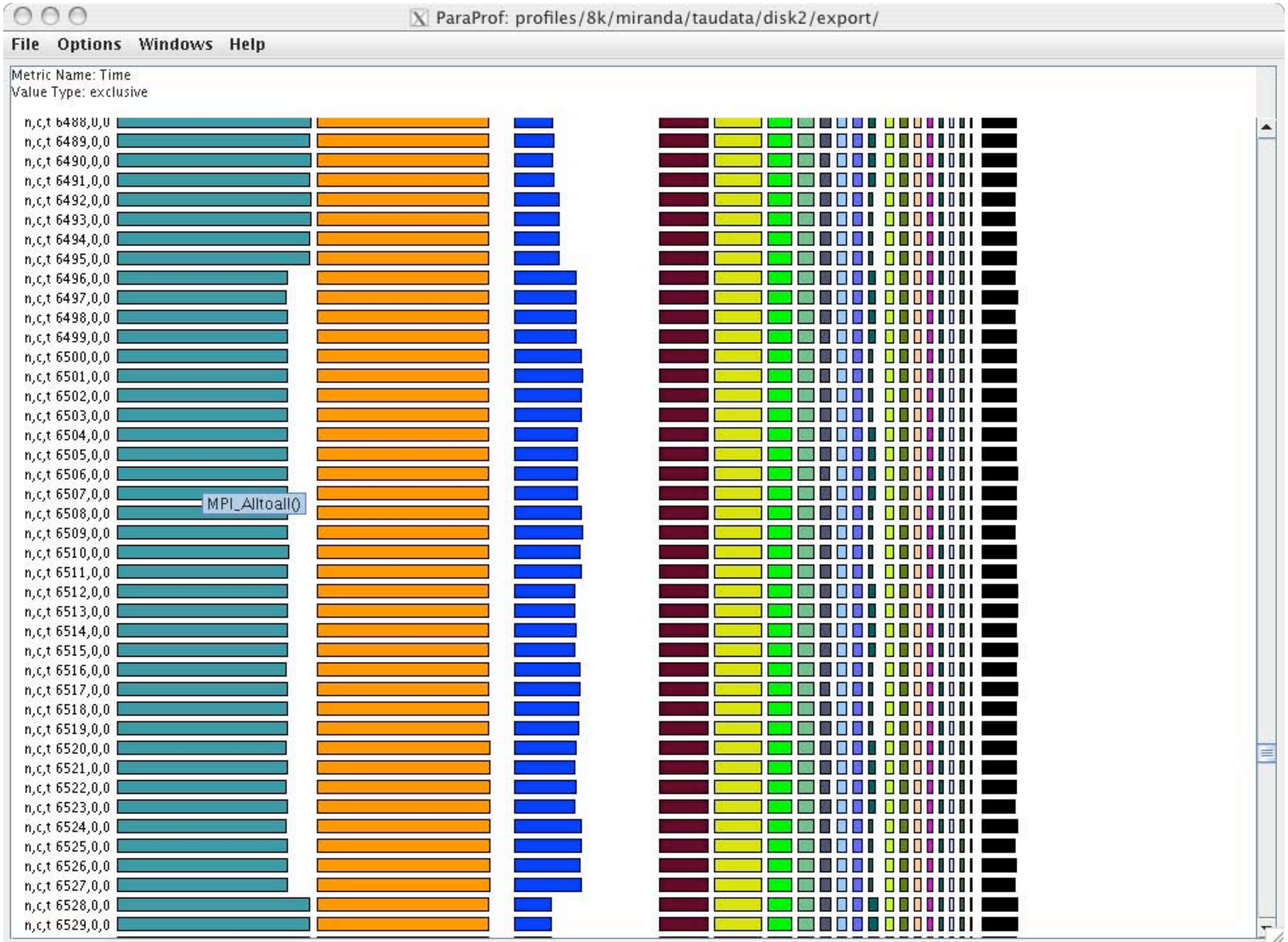
Paraprof Scalable Histogram View



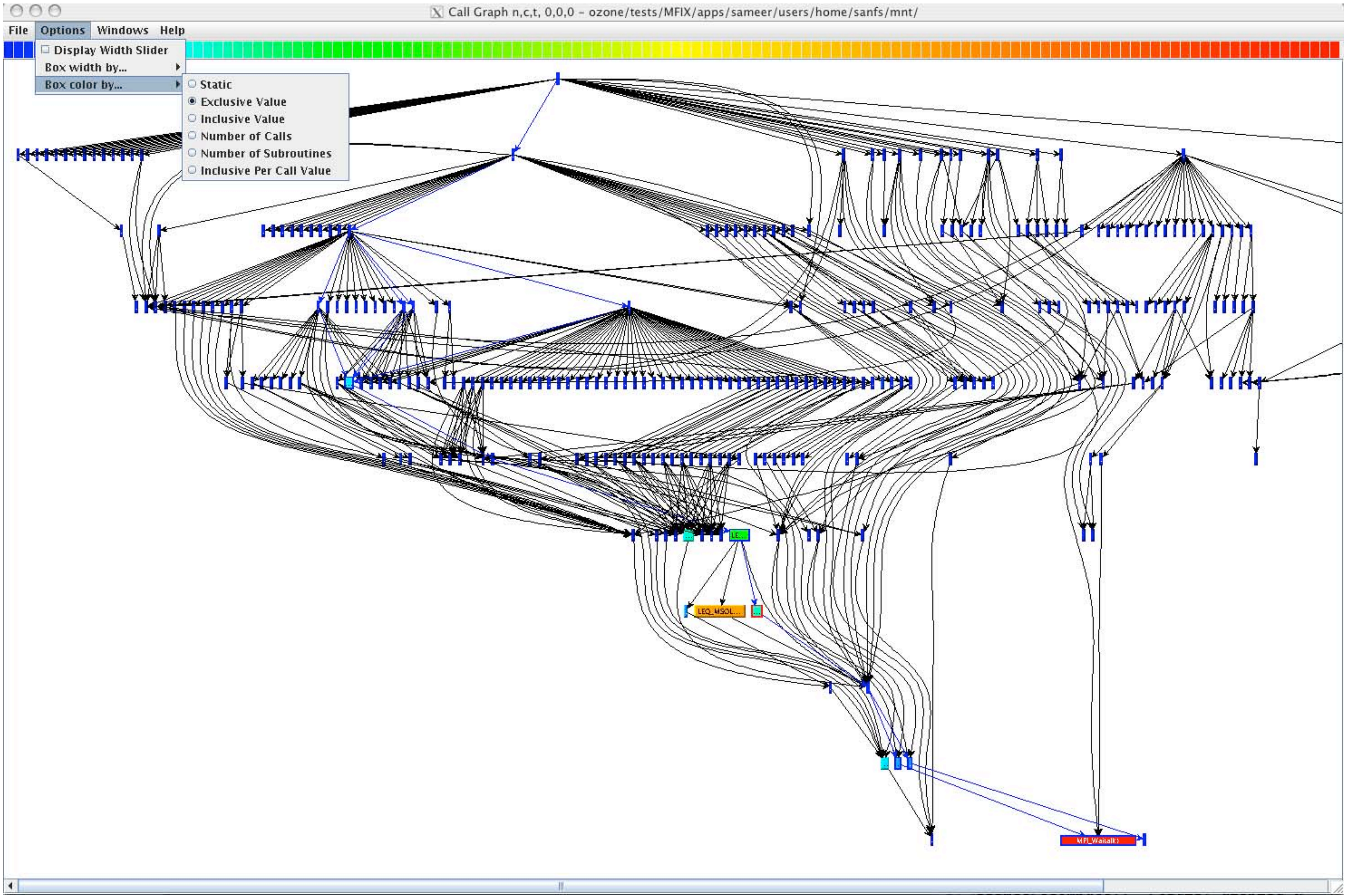
MPI_Barrier Histogram over 16K cpus of BG/L



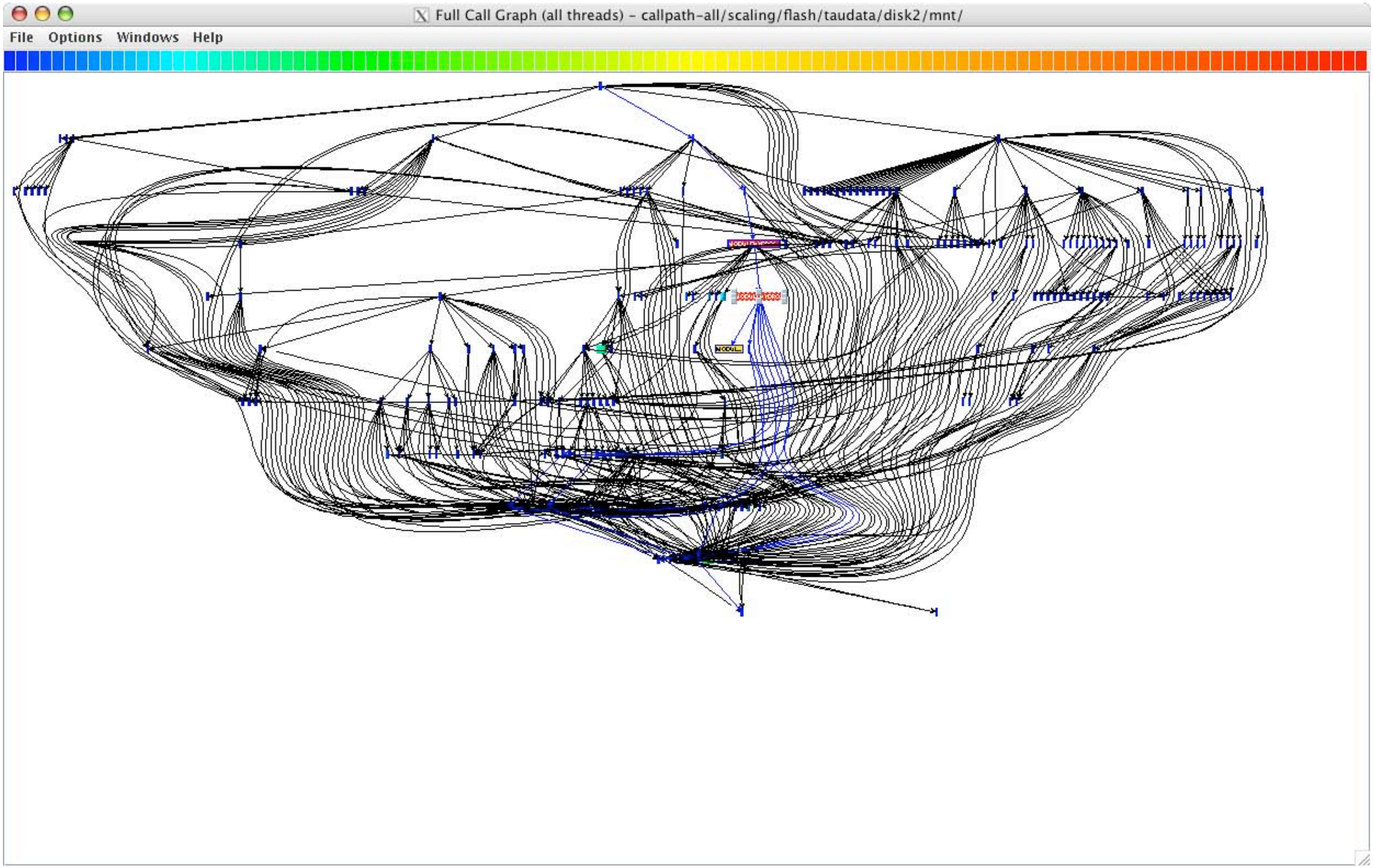
Paraprof Profile Browser



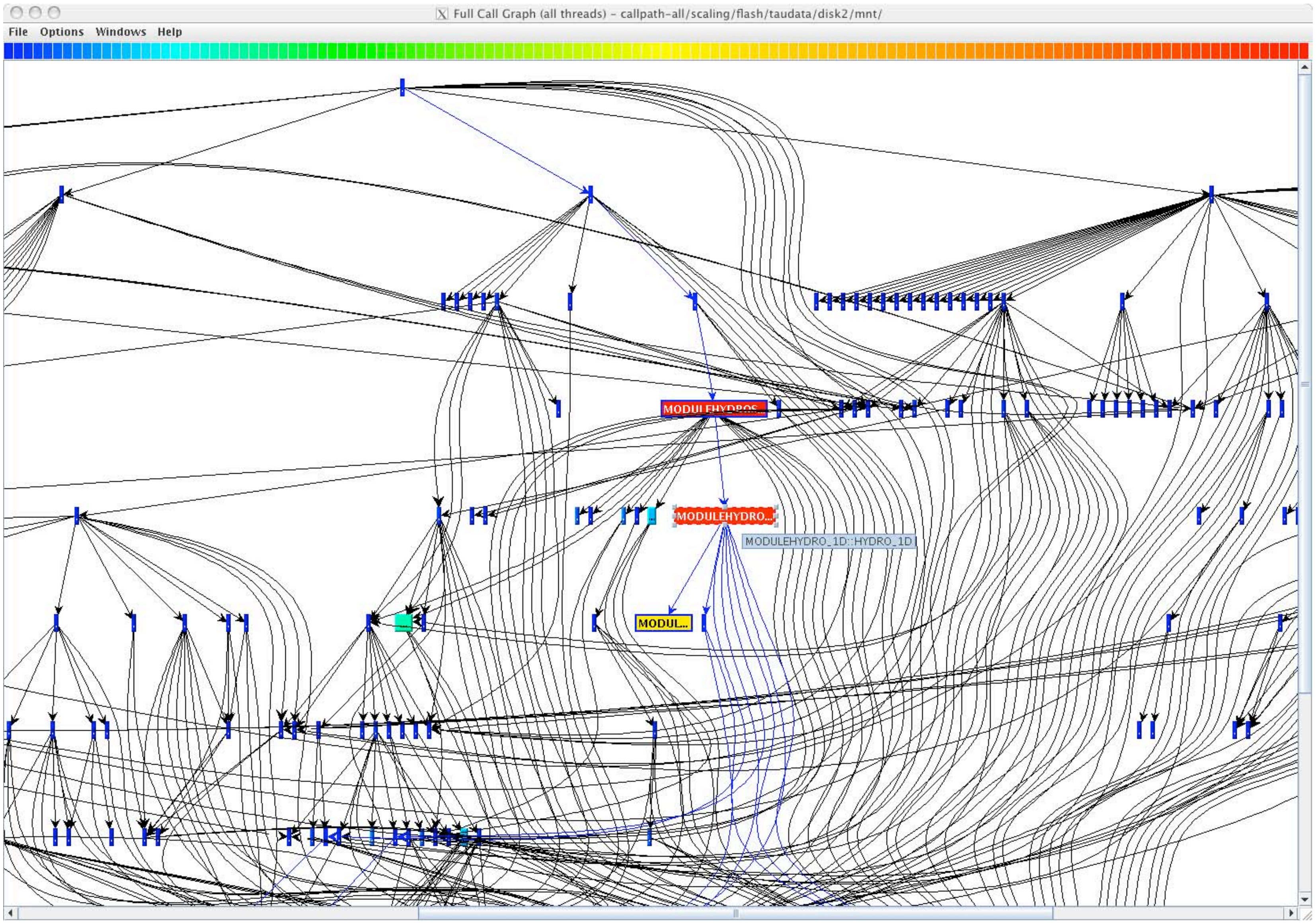
Paraprof – Full Callgraph View



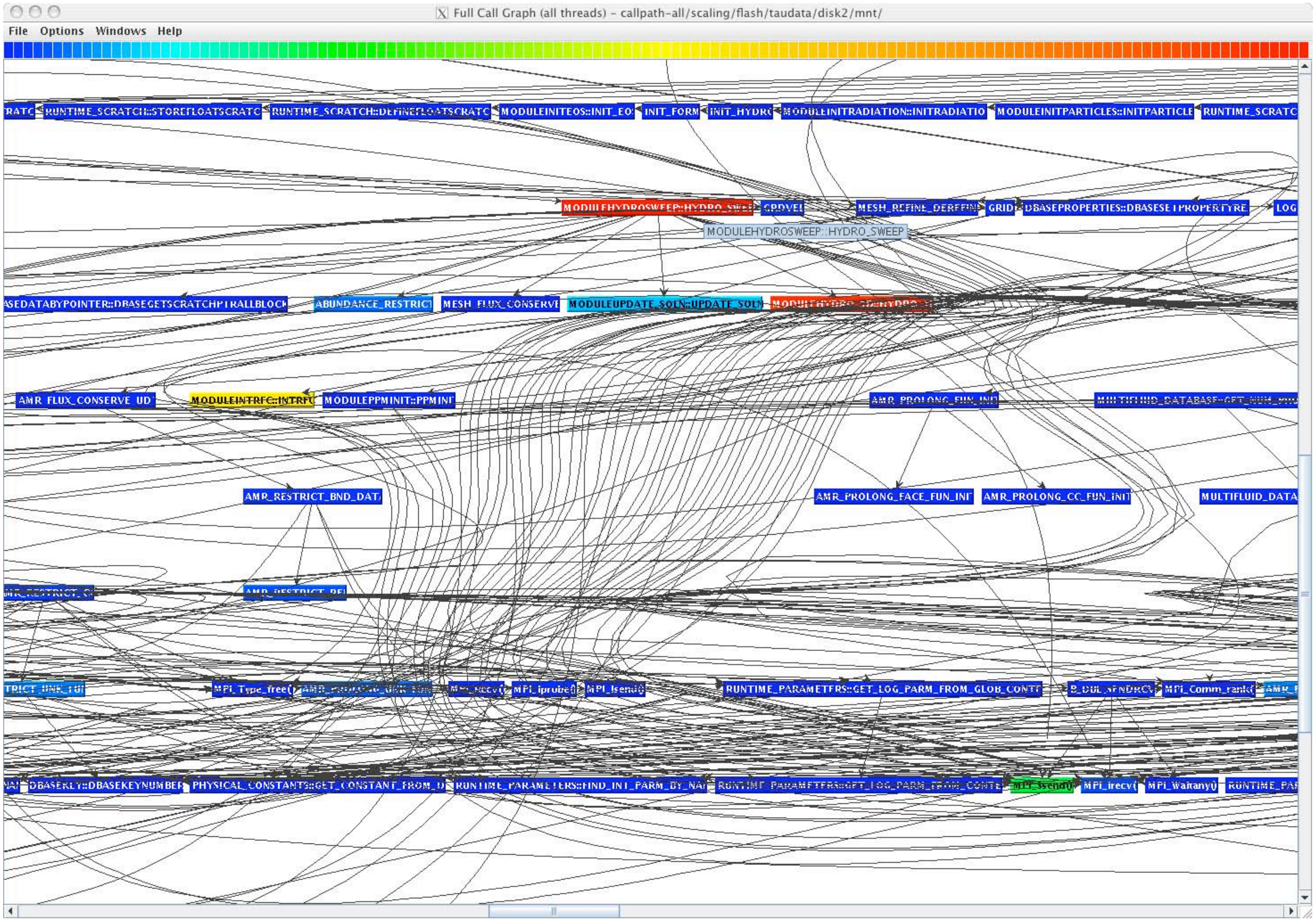
Paraprof – Highlight Callpaths



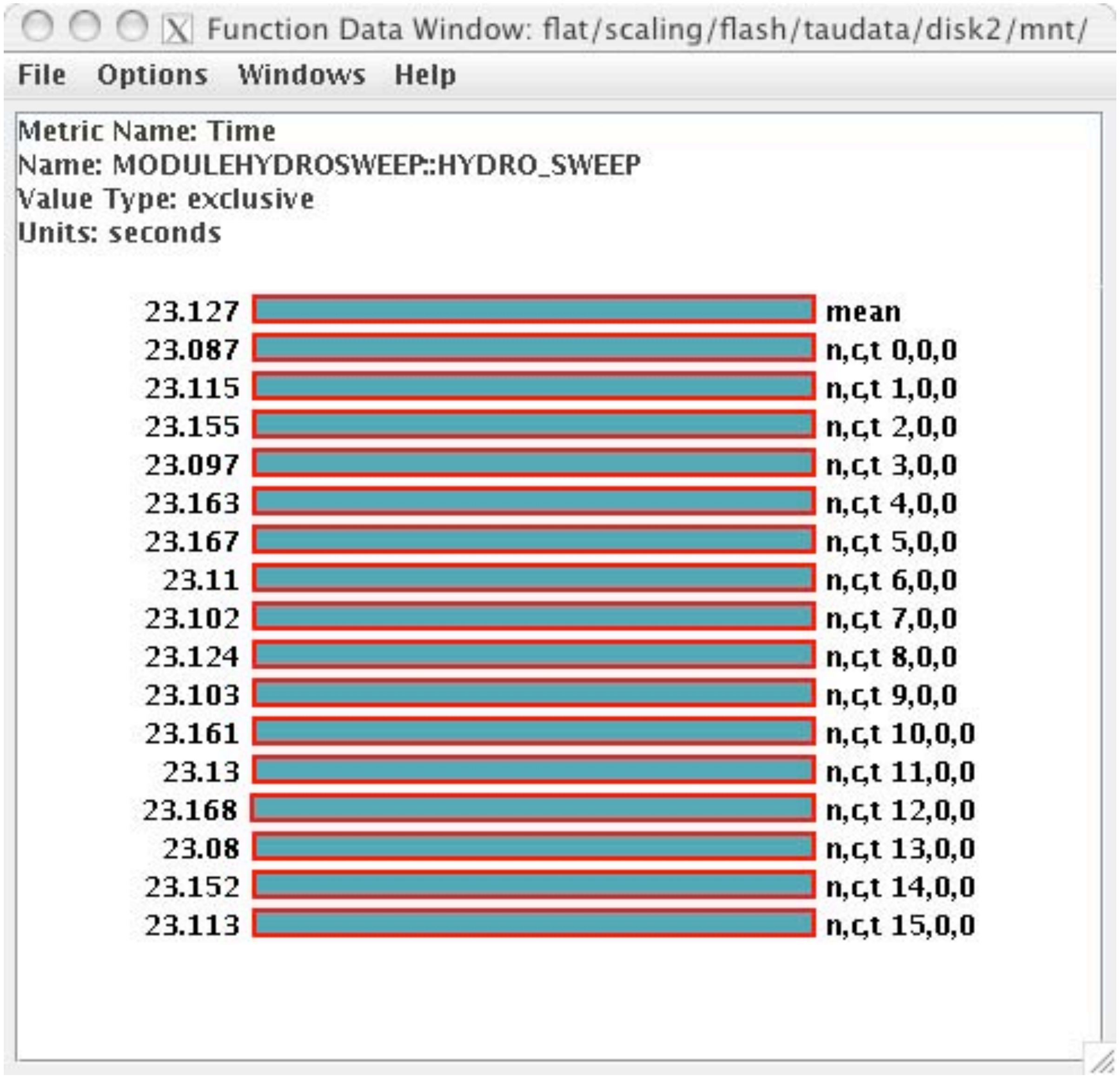
Paraprof – Callgraph View (Zoom In +/Out -)



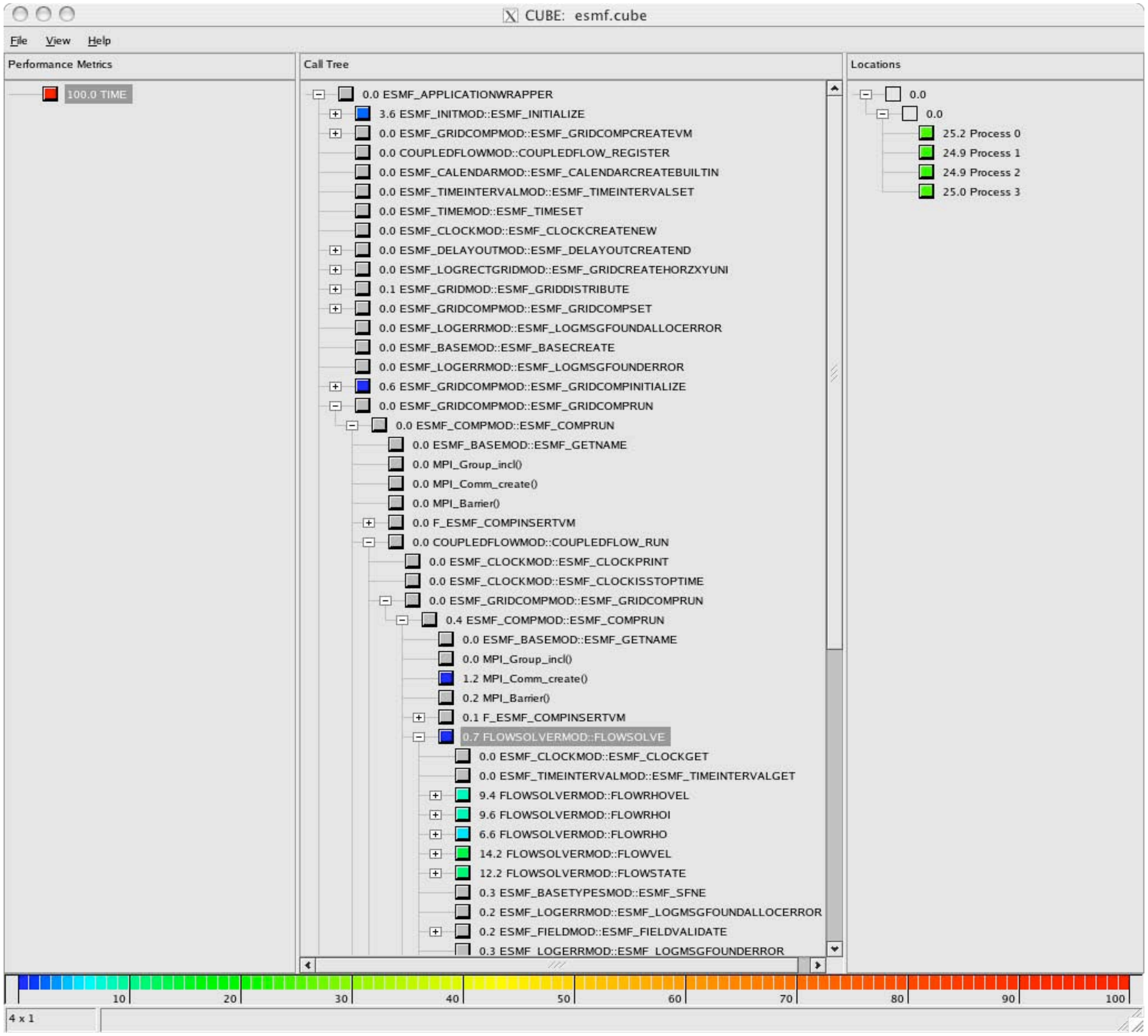
Paraprof – Callgraph View (Zoom In +/Out -)



Paraprof - Function Data Window



KOJAK's CUBE [UTK, FZJ] Browser



Linux Kernel Profiling using TAU

- Identifying points in kernel source for instrumentation
- Developing TAU's kernel profiling API
- Kernel compiled with TAU instrumentation
- Maintains per process performance data for each kernel routine
- Performance data accessible via /proc filesystem
- Instrumented application maintains data in userspace
- Performance data from application and kernel merged at program termination

Kernel Profiling Issues for IBM BlueGene/L

- I/O node kernel - Linux kernel approach
- Compute node kernel:
  - No daemon processes
  - Single address space
    - Single performance database & callstack across user/kernel
  - Keeps track of one process only (optimization)
  - Instrumented compute node kernel

TAU Performance System Status (v 2.14.2.1)

- Computing platforms (selected)
  - IBM BGL, AIX, pSeries Linux, SGI Origin, Cray RedStorm, T3E / SV-1 / X1, HP (Compaq) SC (Tru64), Sun, Hitachi SR8000, NEC SX-5/6, Linux clusters (IA-32/64, Alpha, PPC, PA-RISC, Power, Opteron), Apple (G4/5, OS X), Windows,…
- Programming languages
  - C, C++, Fortran 77/90/95, HPF, Java, OpenMP, Python
- Thread libraries
  - pthreads, SGI sproc, Java,Windows, OpenMP
- Compilers (selected)
  - IBM, Intel, Intel KAI, PGI, GNU, Fujitsu, Sun, NAG, Microsoft, SGI, Cray, HP, NEC, Absoft, Lahey

Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science contracts
  - University of Utah DOE ASCI Level 1 sub-contract
  - DOE ASC/NNSA Level 3 contract
- NSF Software and Tools for High-End Computing Grant
- Research Centre Juelich
  - John von Neumann Institute for Computing
  - Dr. Bernd Mohr
- Los Alamos National Laboratory

Early Results on Blue Gene at SDSC

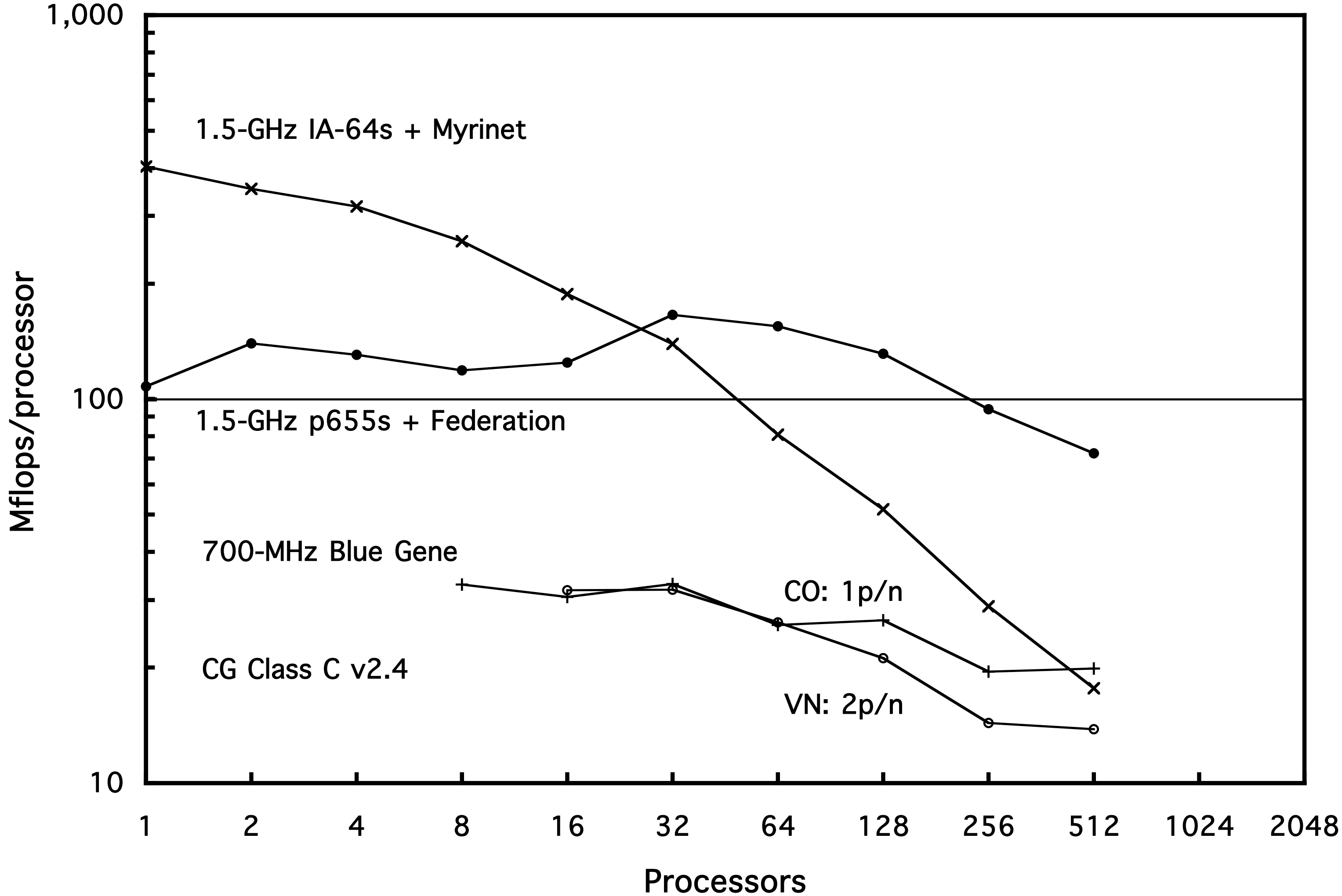
Wayne Pfeiffer

February 24, 2005

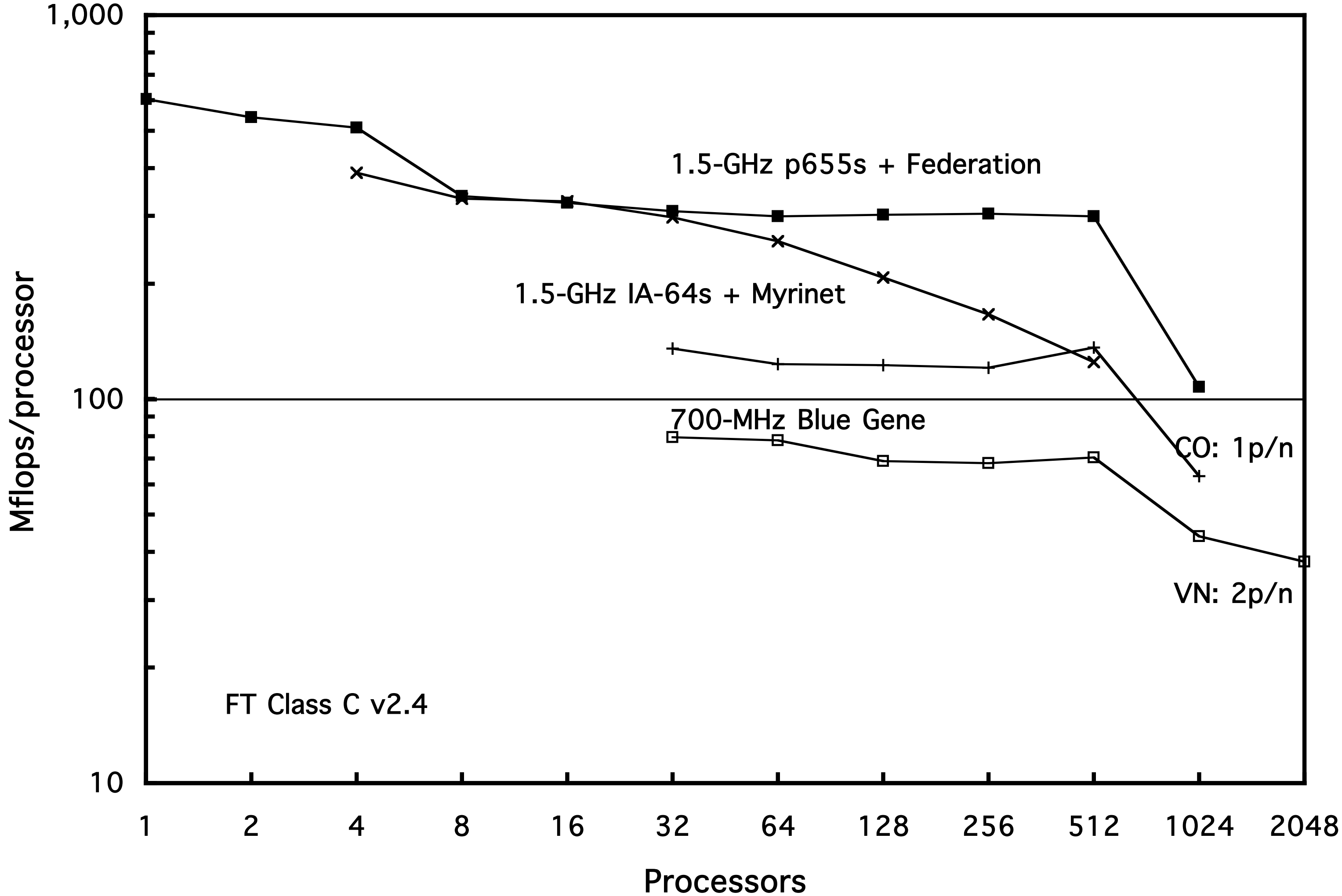
Strong scaling results were obtained on Blue Gene for four NPB 2.4 C kernels & NAMD 2.5

- **NPB kernels were compiled with no changes**
  - Options were -O3 -qarch=440d for CG, MG, & LU
  - Options were -O5 -qnopia -qarch=440d for FT
- **NAMD required numerous changes to compile**
  - Changes were made by student from UIUC visiting IBM
  - Options were -O -qarch=440 (i.e., no optimization to avoid NaNs)
- **mpirun (with Driver 521 for NPBs & 480 for NAMD) used**
  - -partition to specify block (still generally necessary for VN mode)
  - -mode CO & -mode VN for comparison
  - -env "BGLMPI_EAGER=128" for NAMD runs
- **Results were compared with previous ones on**
  - DataStar with 1.5-GHz p655s + Federation
  - TeraGrid with 1.5-GHz IA-64s + Myrinet

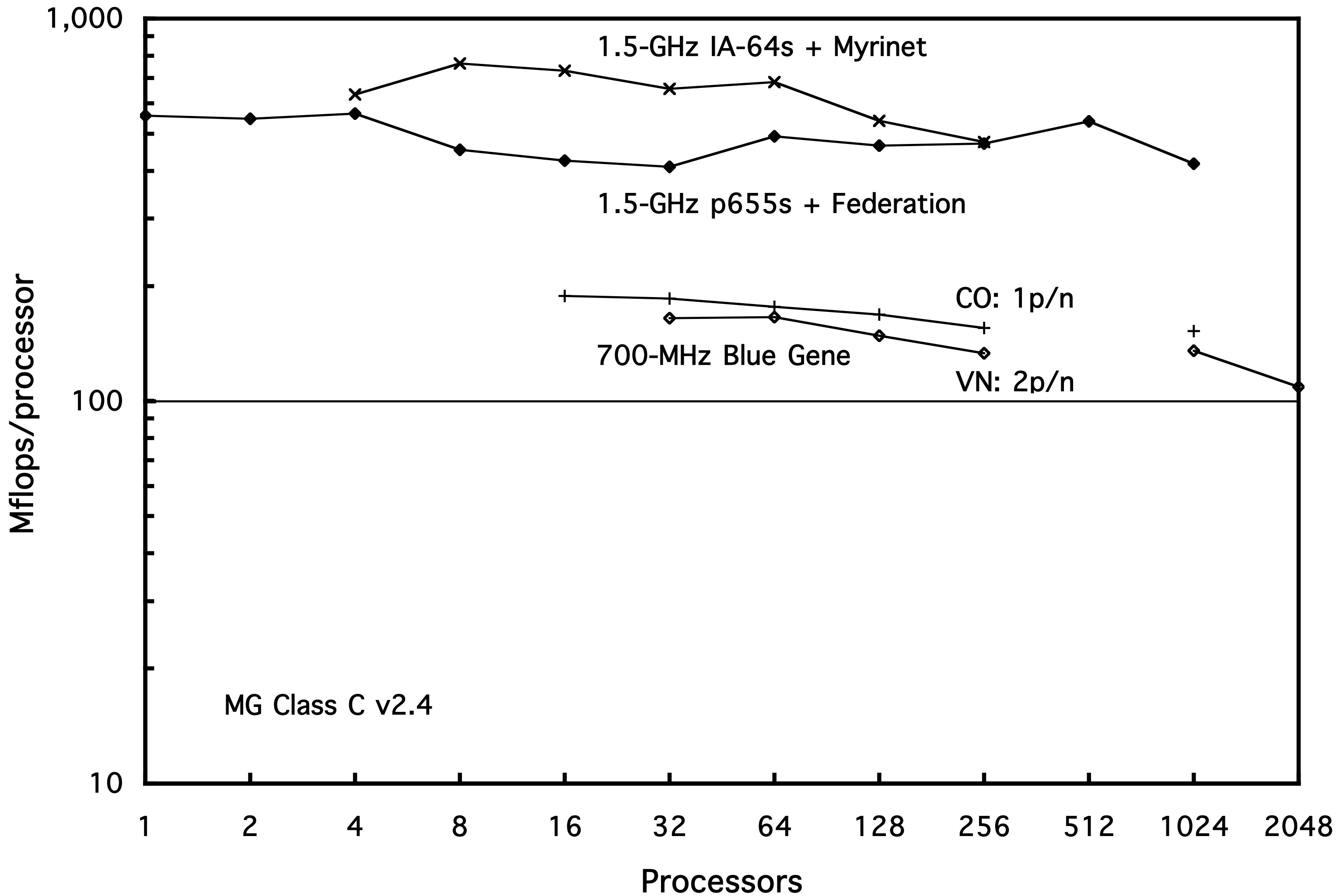
CG scales to 128p or so on BG & p655s, but badly on IA-64s; VN mode is much worse than CO mode (per p) for ≥128p; BG is slow



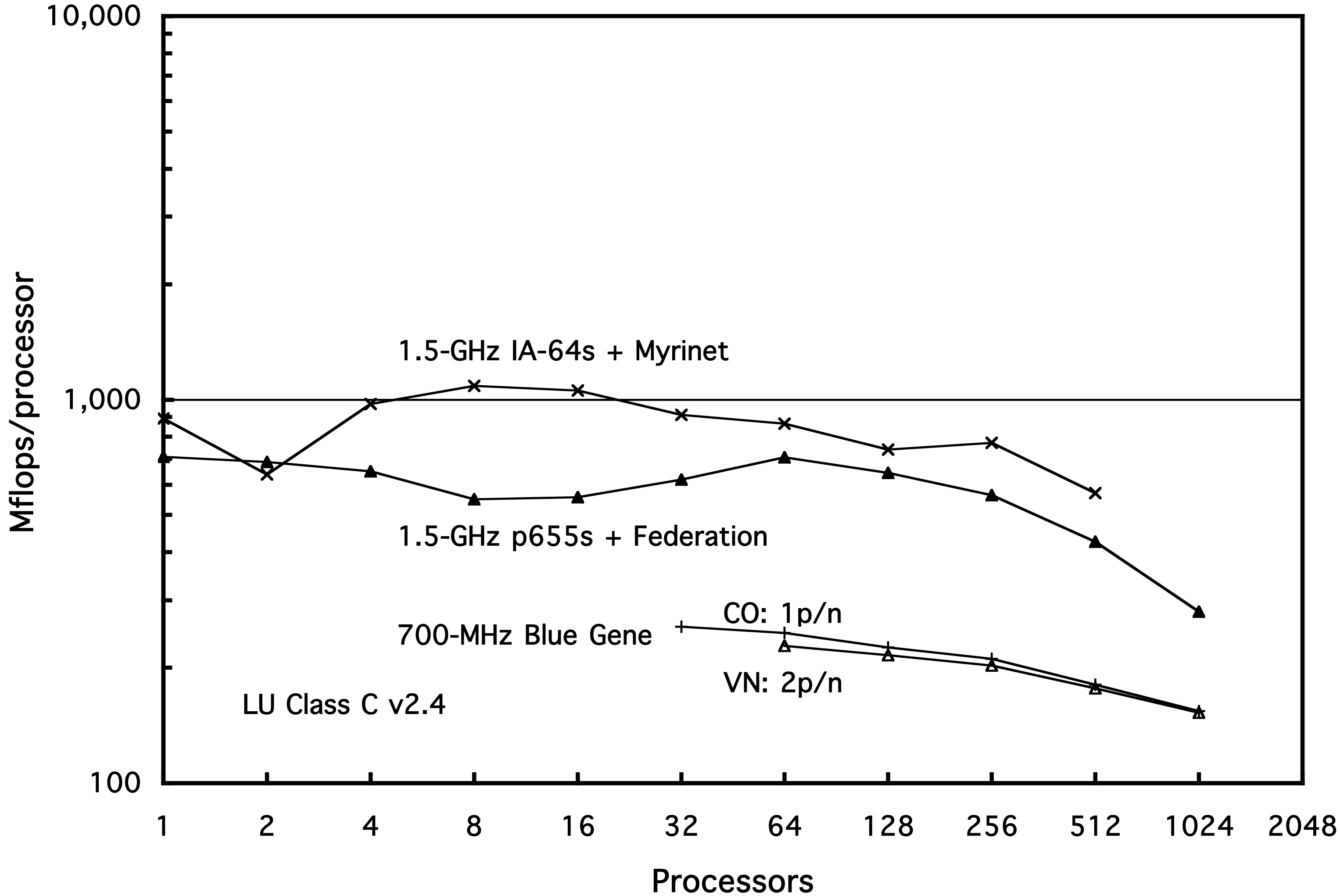
FT scales well to 512p on p655s & BG, but not on IA-64s; VN mode is much worse than CO mode (per p) for all p



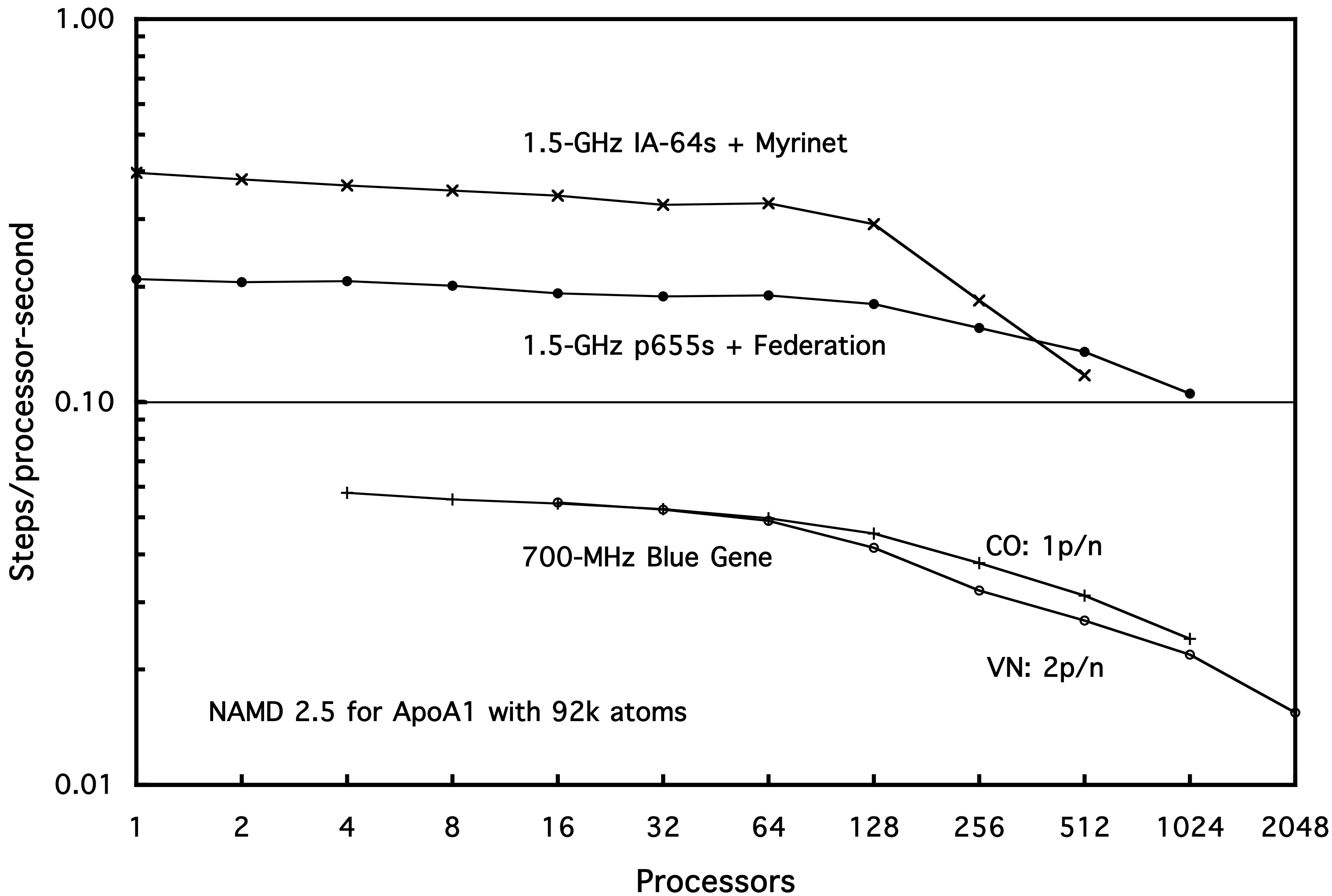
MG scales well on all machines; 512p case fails on BG & IA-64s; VN mode is moderately worse than CO mode (per p) for all p



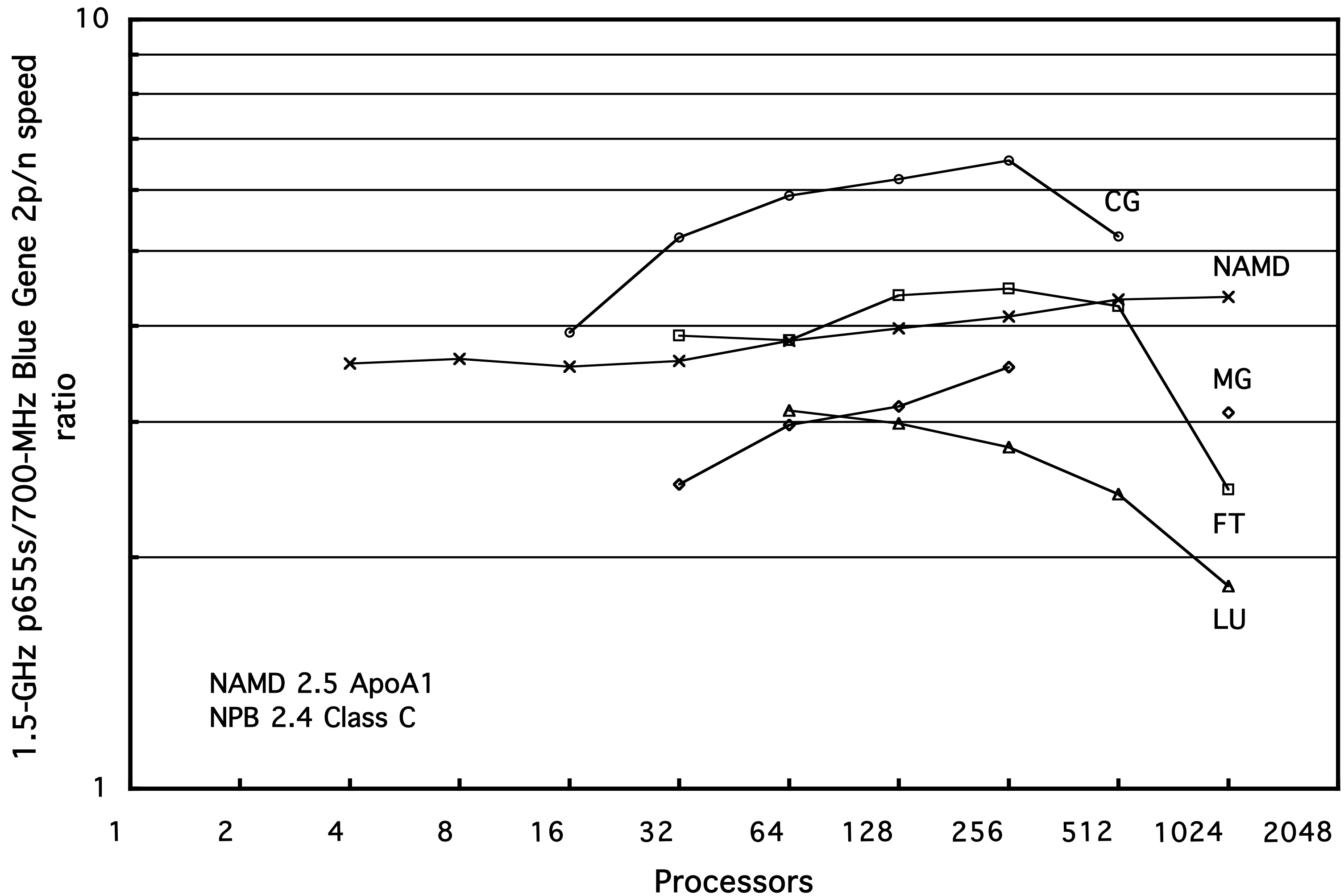
LU scales well to 1024p on BG & 512p on p655s & IA-64s; VN mode is about the same as CO mode (per p) for all p



For NAMD, so far, BG doesn't scale as well as p655s; VN mode is only a little worse than CO mode (per p)



Speed of p655s compared to Blue Gene with VN mode is highest for CG & lowest for LU



Changes in mpirun settings were investigated for NPB 2.4 C kernels

- **No explicit -partition generally faster for ≤256p (only in CO mode now)**
  - Faster with no -partition for all kernels on 64p, 128p, & 256p (presumably because more links are available)
    - by 1.12x to 1.14x for FT & 1.05x to 1.09x for CG
  - About the same for other processor counts, except
  - Slower with no -partition by 0.96x for CG on 32p (?)
- **-connect TORUS generally faster than -connect MESH (only if -partition not specified & only in CO mode now)**
  - Faster with TORUS by 1.11x to 1.16x for FT on 64p to 1024p
  - Slightly faster with TORUS for CG & MG on most processor counts
  - No difference for LU
  - Slower with TORUS by 0.94x for CG on 32p (?)

Changes in driver & compiler flags were also investigated for NPB 2.4 C kernels

- **Driver 521 vs 480 speed the same except**
  - Faster with 521 by 1.15x for FT on half rack (512p) in CO mode
  - Faster with 521 by 1.09x for FT & 1.05x for MG on half rack (1024p) in VN mode
- **-O5 -qnoipa vs -O3 speed the same except for FT**
  - Faster with -O5 -qnoipa by 1.03x for FT on 32p to 512p in CO mode
  - Faster with -O5 -qnoipa by 1.05x for FT on 32 & 64p in VN mode
  - Slower with -O5 -qnoipa by 0.95x for FT on 2048p in VN mode

Other codes are being ported &/or tested

- **CPMD: quantum molecular dynamics**
  - Running executable from IBM
  - Need full rack to have enough memory for problem of interest
- **DOT: protein docking**
  - Tracking down NaNs during execution
- **DNSMSP: 3-D turbulence**
  - Need to change FFT algorithm from 1-D to 2-D decomposition to use less memory & scale better
- **ENZO: 3-D cosmology**
  - Need to eliminate some scaling bottlenecks to run problems of interest
  - Need fast, parallel I/O
- **SPECFEM3D: 3-D seismic wave propagation**
  - Compiles with -O5 -qarch=440d
  - Have trouble fitting in memory of single rack
  - Need fast, parallel I/O

System software priorities for SDSC

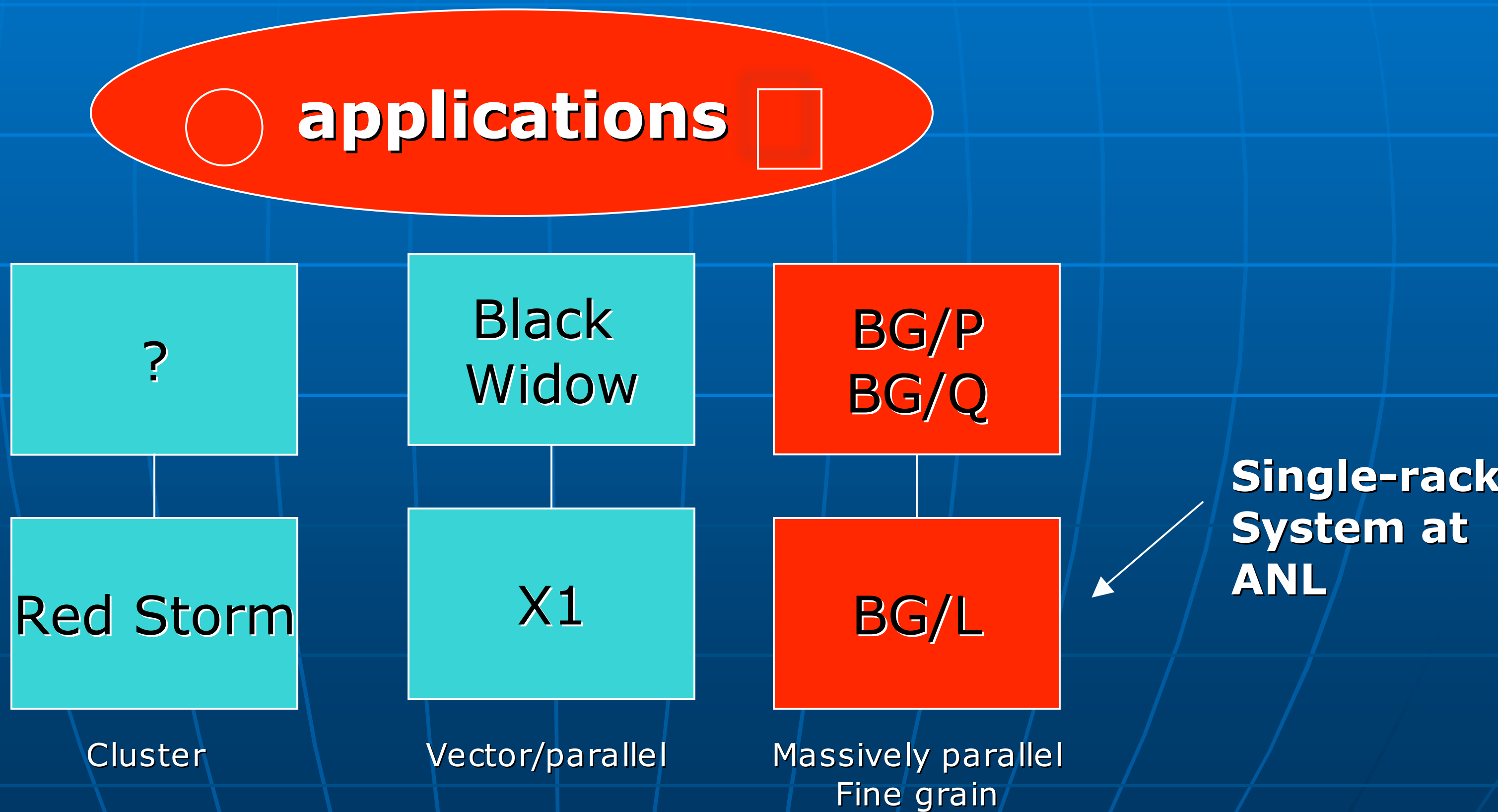
- mpirun with anomalies and hangs fixed
- LoadLeveler for batch job submission
- GPFS for fast, parallel I/O
- HPC Toolkit for performance analysis
- TotalView debugger

Initial Application Porting

Andrew Siegel
Katherine Riley
Argonne National Laboratory

Project Goals

- How will applications map onto different petaflop architectures?



Benchmarking BG/L

- Three layers of tests
  - Microbenchmarks
    - STREAM, mpptest, Euroben, Parkbench lmbench, SKaMPI, IO/ Tile test, HPC Challenge, Vector add and compiler options
  - Application kernel benchmarks
    - Petsc FUN3D, sPPM, UMT2000, NAS PB-MPI
- Web site constanly updated
  www-unix.mcs.anl.gov/~gropp/projects/parallel/BGL/index.htm

Benchmarking, cont.

- Application benchmarks
  - POP (Los Alamos Ocean Simulation)
  - QMC (monte carlo nucleonic forces)
  - Flash (Astrophyics -- hydro, burning, mhd, gravity)
  - Nek (Biological fluids – spectral element cfd)
  - Nimrod (Fusion – toroidal geometry)
  - pNeo (Nueroscience – Huxley nueron model)
  - Gyro (Plasma microturbulence)
  - IP
  - QCD (Lattice QCD)
  - Decartes, Ash, QGMG pending ..

Applications not Ported

- Require MIMD
  - Coupled ocean-atmosphere model
  - Coupled neutronics-hydro reactor model
- Codes with commercial components
  - e.g. Star-CD common for multiphase flow
- Codes with drivers written in Python

Application porting strategy

- Each application scientist gets 32-node dedicated partition for porting/tuning.
- Nightly full-rack reservations for bigger runs
- Mailing list with many contributors to help with porting, tuning, debugging issues.

Application expectations

- Current 1-rack system likely to do problems 1-2X size of our current Pentium/Myrinet Cluster
  - 1024 vs. 350 nodes
  - 2-3X performance / node on Pentium
  - Better scalability on BG/L
- Goal: scale to 10-20 rack system

Performance Measurements

Performance Matrix

app\metric	kernel	Communication pattern	Comp rate	Weak Scaling?	Threshold Memory	i/o	Primary Scalability bottlenecks
Flash	▪ Explicit hydro ▪ Multigrid ▪ SODE	dynamic ▪ Nearest Nghbr ▪ Global Ops	4/1	yes + αP^2	128Mb	O[1 Tb]	▪ Global block redistribution ▪ AMR multigrid
Nek5	Matrix-matrix product	fixed ▪ Nearest Nghbr ▪ 2 Global Ops	7/1	yes	16Mb	O[1 Tb]	none
QMC	Sparse matrix operations	fixed ▪ Master/slave	10/1	yes	O[Kb]	O[10 Mb]	non-scalable
pNeo	SODE	dynamic ▪ Neighborhood	1/2	yes+ non-trivial topology correction	1Gb	O[1 Mb]	Reduction operations over neighborhood
Columbus	Eigenvalue problem (Davidson method)	fixed ▪ Neighborhood	5/1	no	?	O[?]	?

Definitions
- *Comp Rate*: Estimated ratio of local work to communication time for RAM=256Mb
- *Weak Scaling:* Yes if ratio computation/comm constant at fixed local work
- *Threshold memory*: Local system memory where communication time = local work time

Application Performance

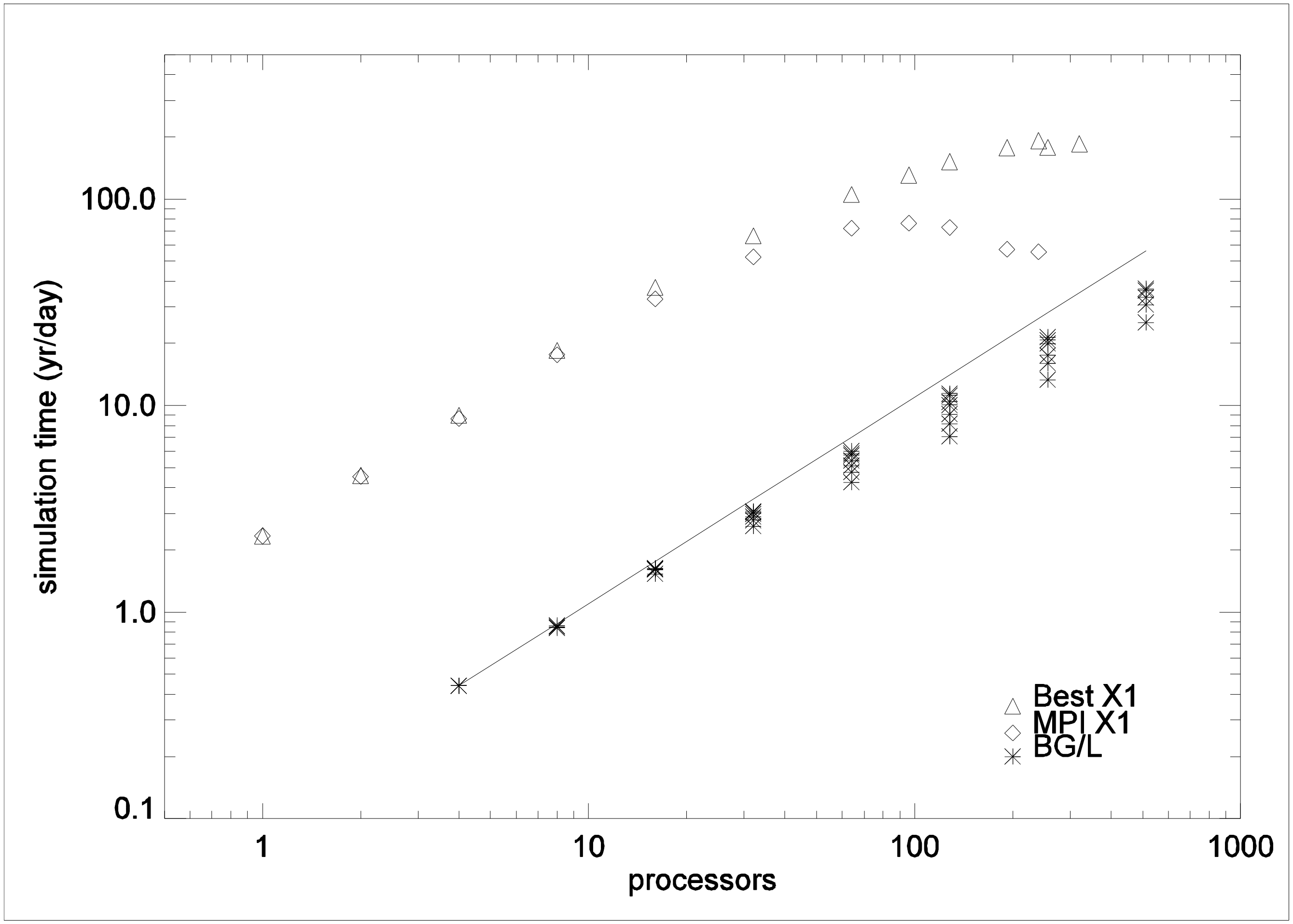
- General observations
  - Porting much easier than expected
    - Most programs have run extensively on NERSC mach
  - Single proc performance on poor end of expectations
    - No use of double-hummer
    - Uncertainty about data alignment issues
    - Loop unrolling limits give larger variations than we typically experience
    - One case of slow math intrinsics (using libm)
    - No essl
    - Addicted to hpmlib feedback to diagnose performance!
    - -qdebug=diagnostic doesn't work on our system

Application Performance

- General Observations, cont.
  - Network performance
    - Appears to be very good compared to what we're used to
    - Extremely reproducible timings
    - Still lots of detailed tests to run
  - VN mode
    - Most applications have at least one interesting problem which can be run with ½ the memory
  - IO
    - Haven't stressed it much at apps level

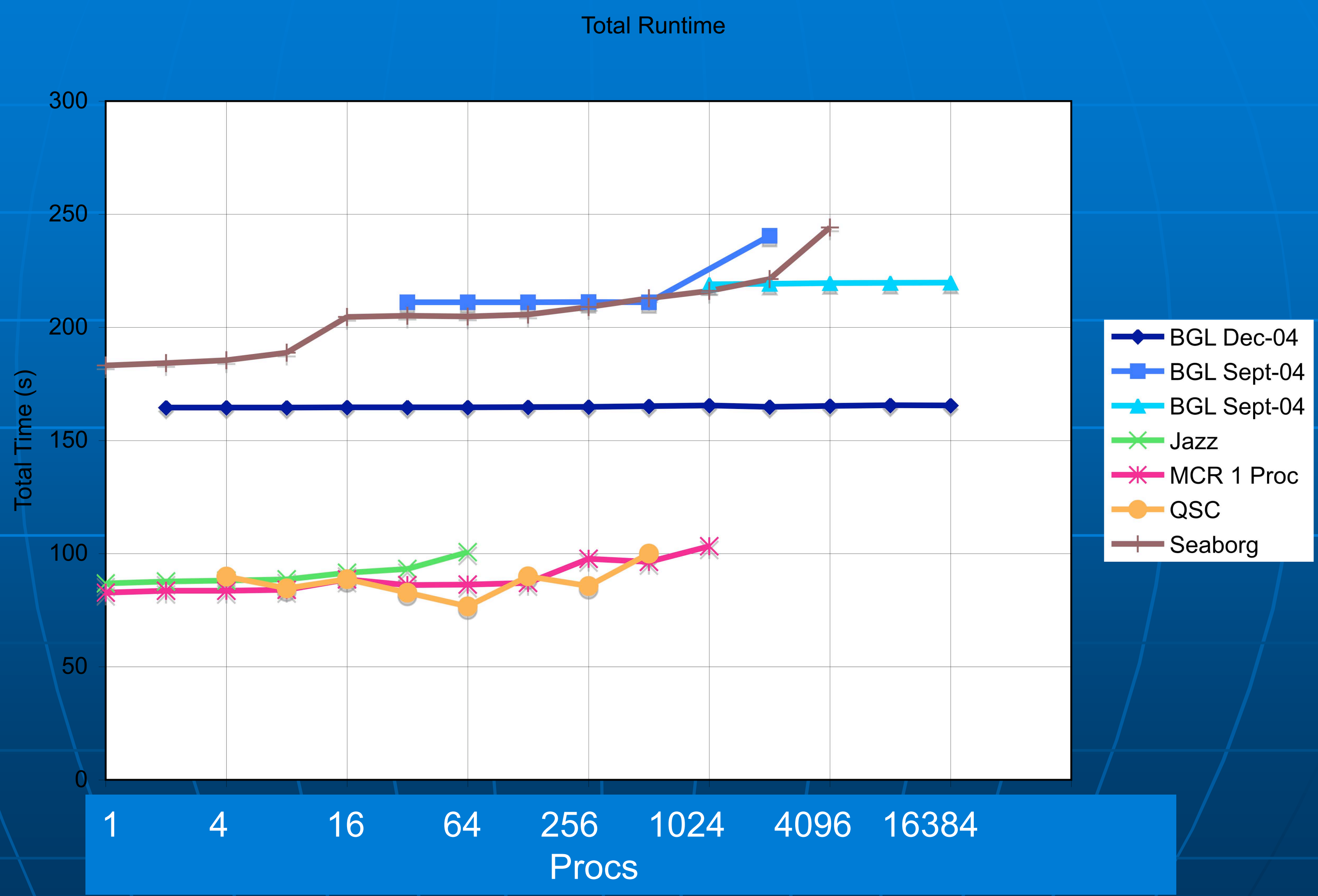
Some preliminary performance

POP Test



simulation time (yr/day)
100.0
10.0
1.0
0.1
1
10
100
1000
processors
Best X1
MPI X1
BG/L

Total Time For 2D Sod



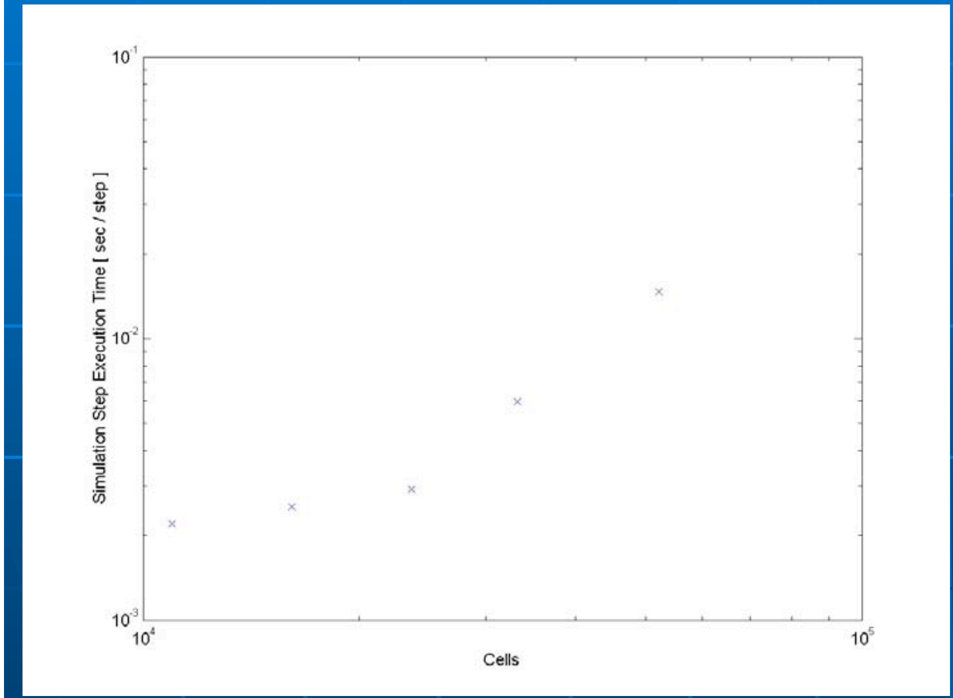
Ported tools/frameworks

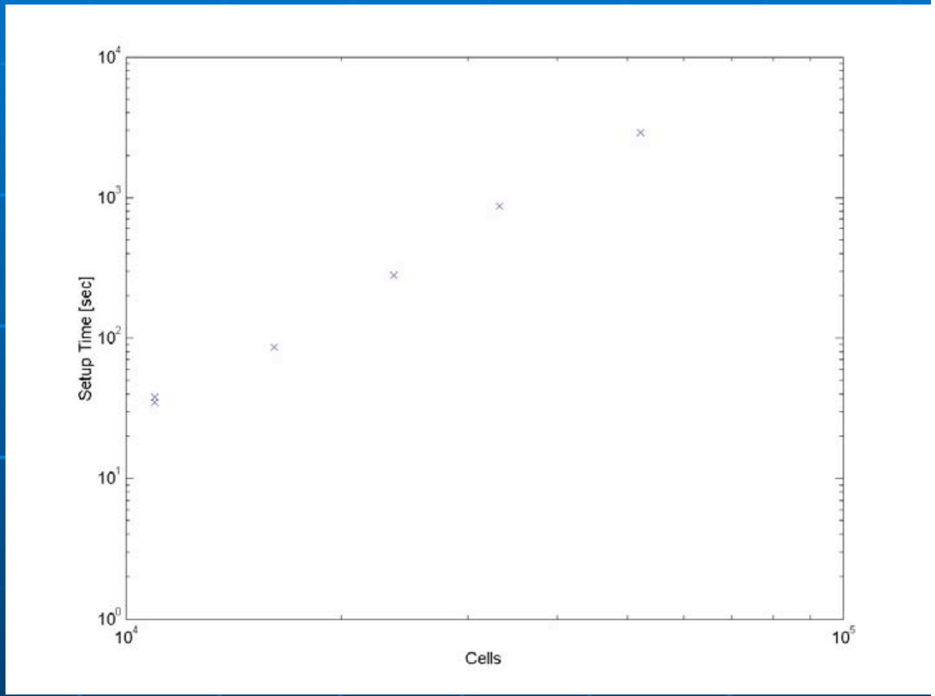
- TAU (U. or Oregon)
- PetsC
- fpmpi
- jumpshot

Summary of Application Needs

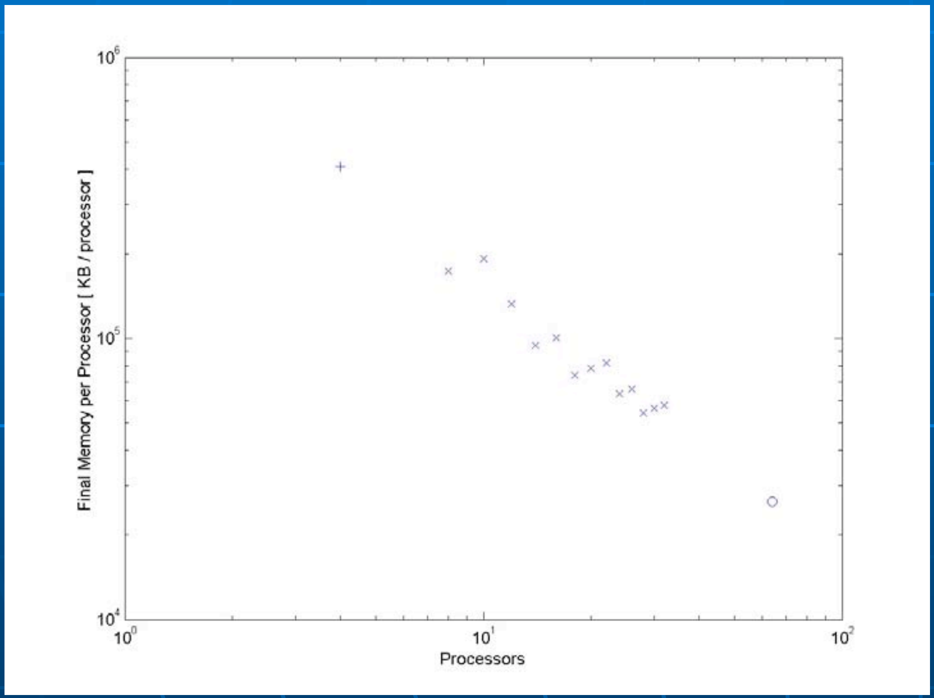
- Compilers
  - Double hummer assembly
  - Report functionality
  - Extended SIMD capabilities
  - Data alignment clarity
- Math Libraries
  - ESSL, mass(v), BLAS
- I/O : mpi i/o
  - hdf5, pNetcdf
- Debugger: gdb
- Profiler: gprof
- Software updates
  - Fixes to mpirun, compiler bugs
- HPM Lib | PAPI
- Stack/overwriting memory
- Better memory diagnostics (TAU?)
- General app requests
  - Dynamic libraries
  - MIMD possibilities
- Double FPU instructions

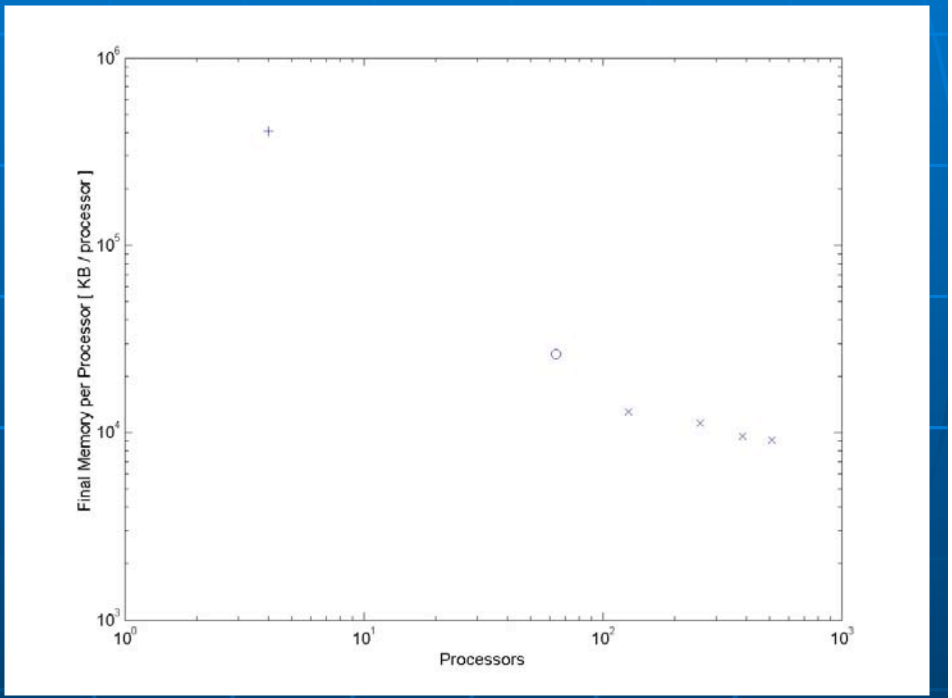
pNeo tests – problem size



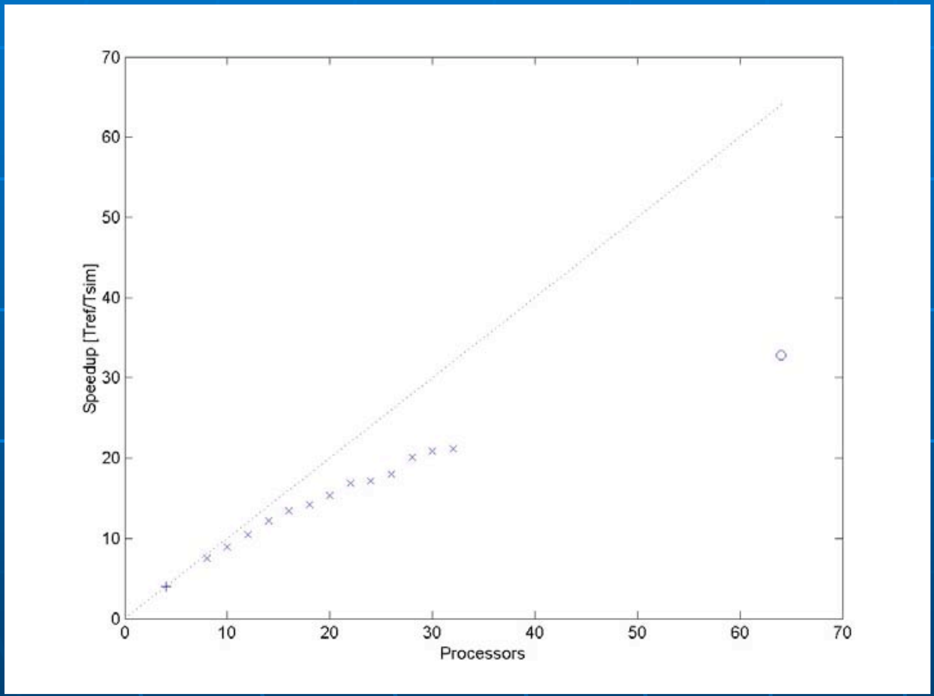


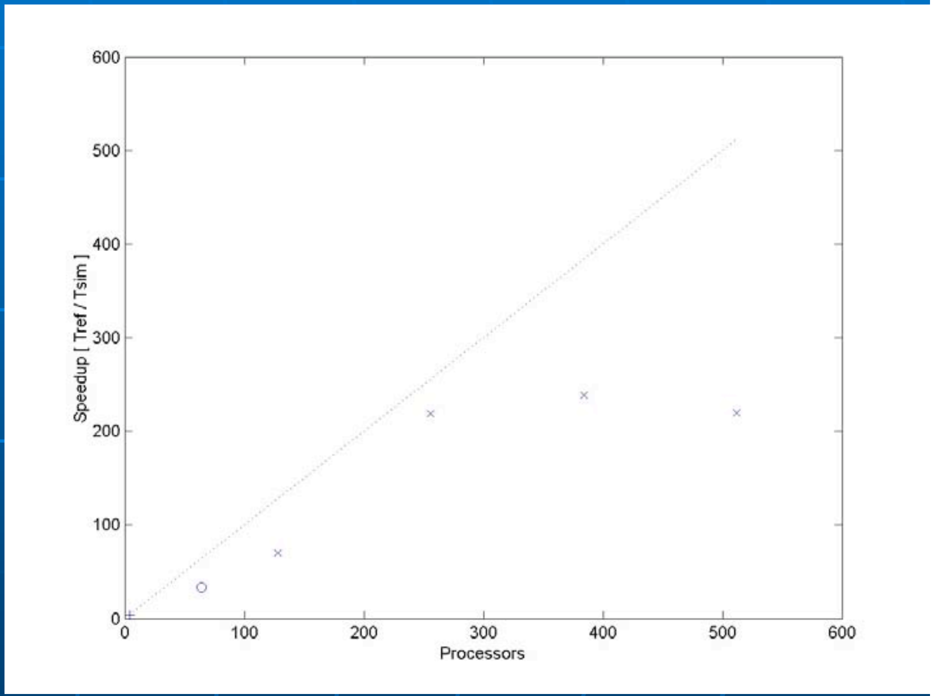
pNeo tests, cont.



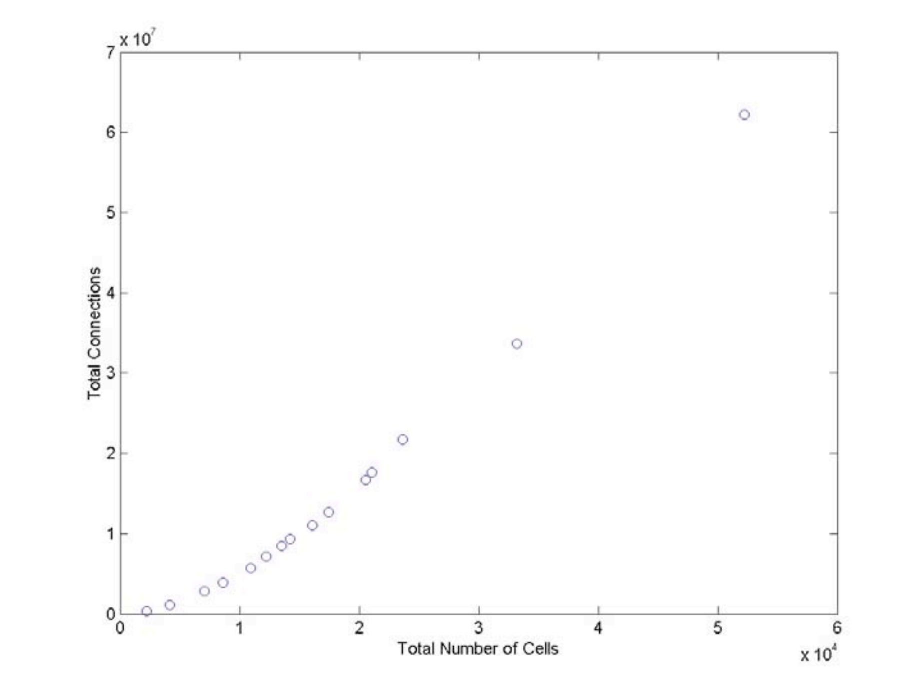


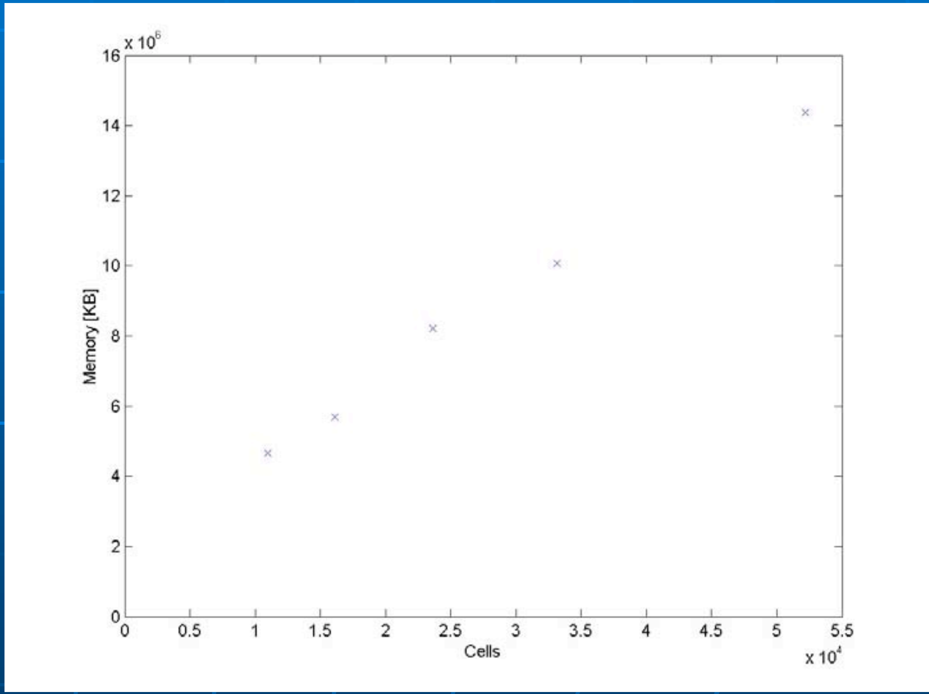
pNeo tests, cont.





pNeo tests, cont.





BlueGene/L: Early Application Scaling Results

Steve Louis and Bronis R. de Supinski
Lawrence Livermore National Laboratory

NNSA ASC Principal Investigator Meeting &
BG/L Consortium System Software Workshop
February 24, 2005

UCRL-PRES-209861

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

Wide Variety of Early Applications on BlueGene/L

Blue Matter (IBM) *
Flash (ANL) *
Miranda (LLNL) *
MM5
Amber7, Amber8
GAMESS
QMC (Caltech)
LJ (Caltech)
PolyCrystal (Caltech)
PMEMD (LBL)
LSMS (ORNL)
NIWS (NISSEI)
HOMME (NCAR) *
Qbox (LLNL)
ddcMD (LLNL)

SAGE (LANL)
SPPM (LLNL)
UMT2K (LLNL)
Sweep3d (LANL)
MDCASK (LLNL)
GP (LLNL)
CPMD (IBM/LLNL) *
TLBE (LBL)
HPCMW (RIST)
ParaDiS (LLNL)
QCD (IBM)*, QCD (BU) *
NAMD
PAM-CRASH (ESI)
Raptor (LLNL) *
Enzo (SDSC)

Successful scaling tests and science runs completed for key ASC codes

- Using IBM Rochester BG/L hardware through SC2004 (many, many thanks to Jim Sexton of IBM for his help)

- Several codes running at scale since January 3, 2005 on Livermore's first 16 racks (soon to be 32)

- ASC has concentrated on scaling up the following codes:
  - ddcMD
  - FEQMD
  - GRASP (SNL)
  - hypre/SMG2K
  - LAMMPS (SNL)
  - MDCASK
  - Miranda
  - ParaDiS
  - Qbox
  - Raptor
  - SPaSM (LANL)
  - sPPM (Benchmark)
  - UMT2K (Benchmark)

Excellent scaling seen on runs up to 16K nodes and 32K processors

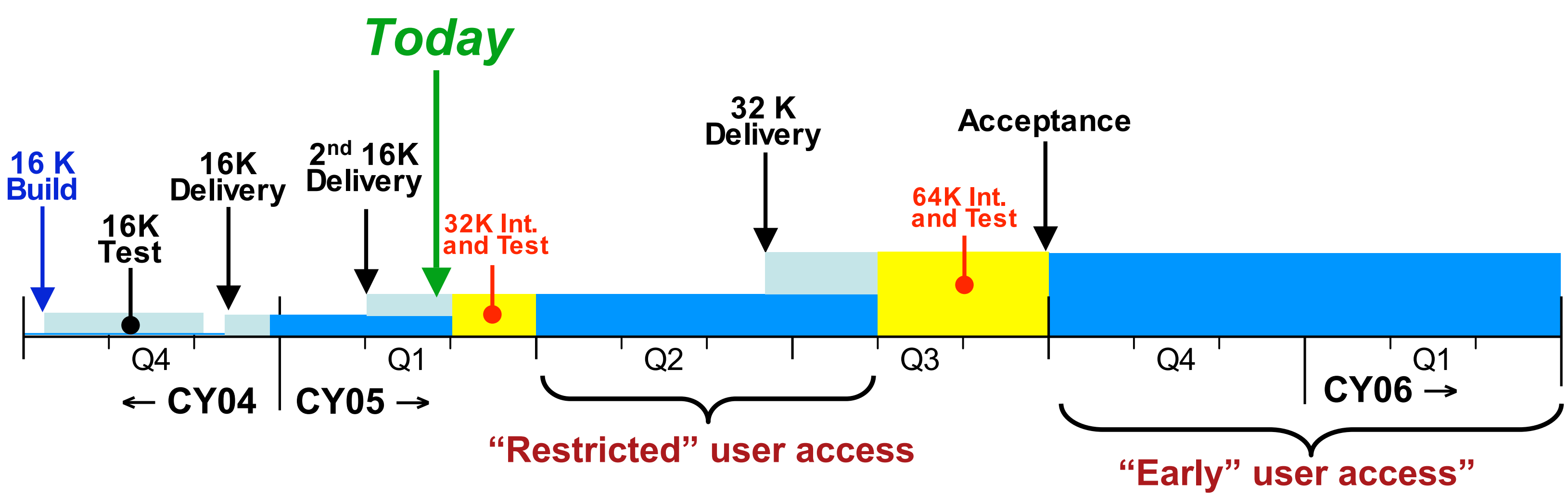
Current Time Line

1. 16 racks delivered in November 2004
   - System tested in December
   - Runs started in January

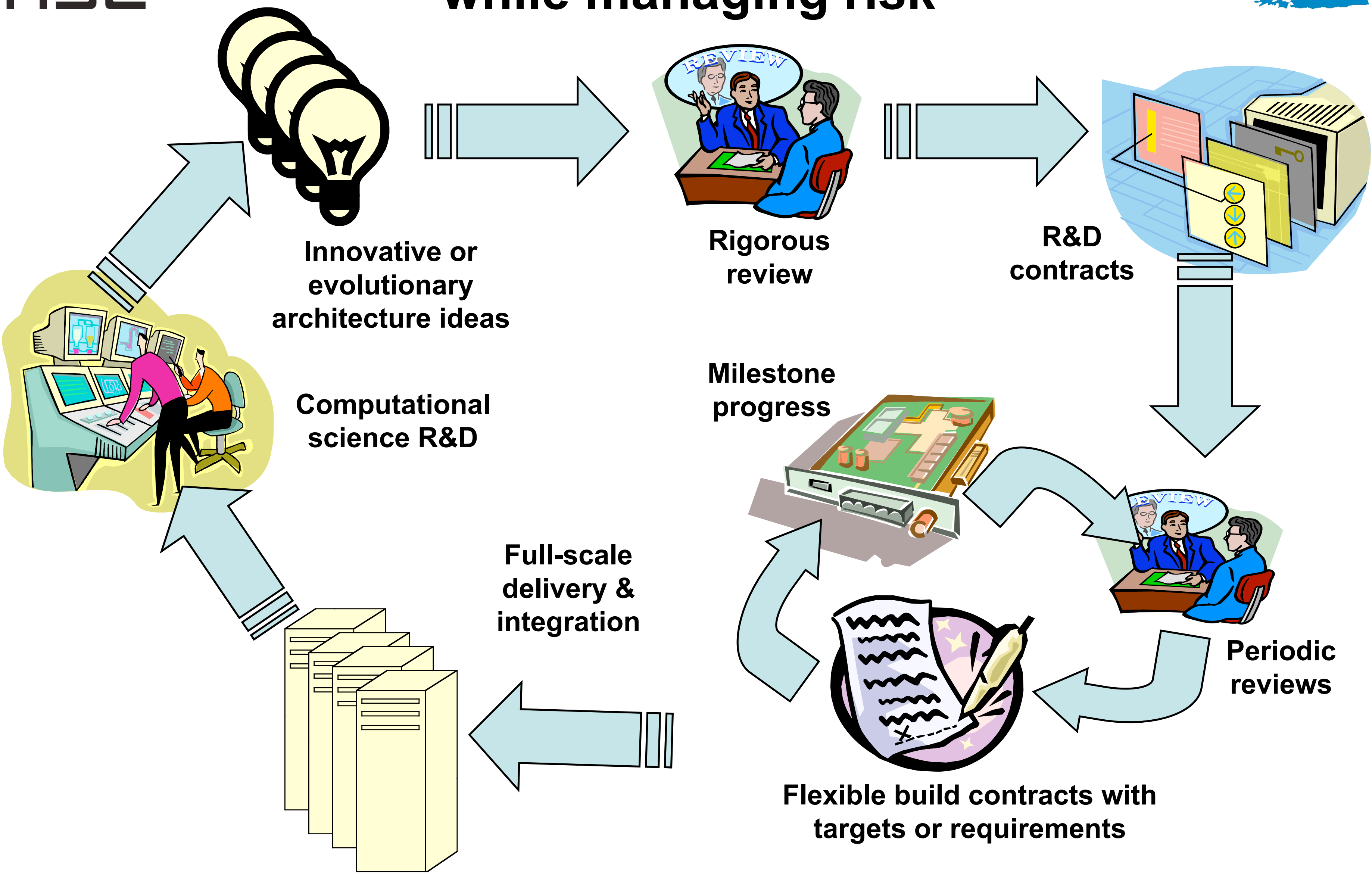
2. 16 additional rack delivery early February
   - 32-rack runs start in April

3. 32 additional rack delivery in mid 2005
   - integration to 64 rack system
   - Machine shake out and test
   - 64-rack scaling/science follows

4. Late 2005 expanded early user access and more science runs

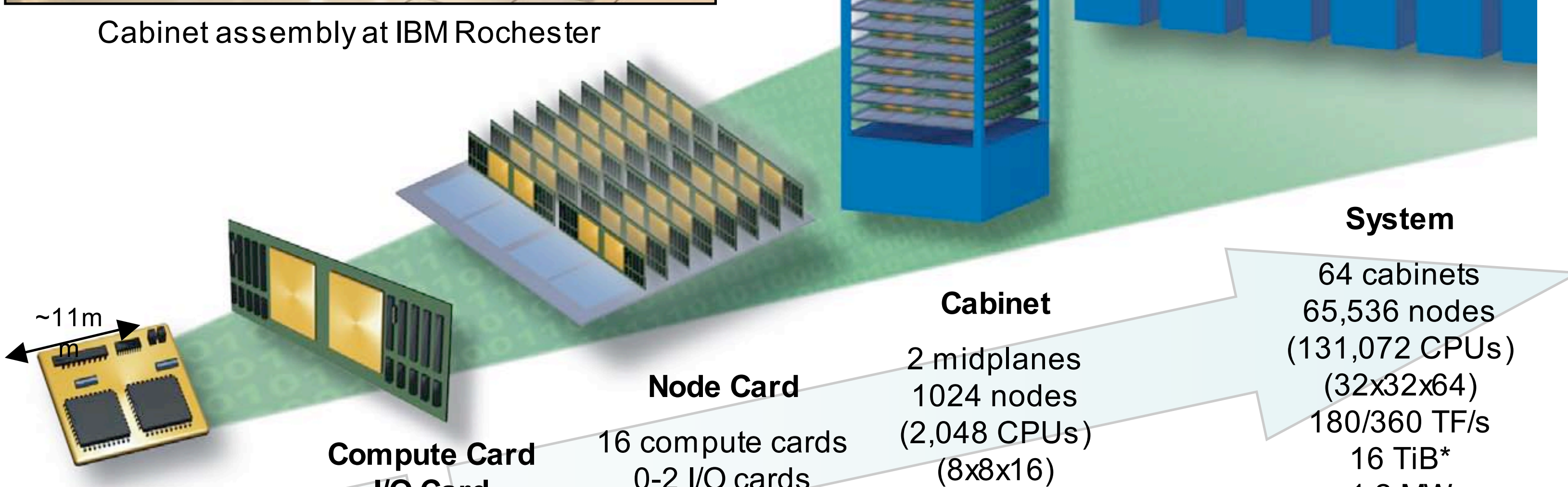


DOE strategic investment in ASC advanced architecture encourages innovative ideas while managing risk



BlueGene/L scales to 360 TF with modified COTS and custom parts

Cabinet assembly at IBM Rochester



Compute Chip

2 processors
2.8/5.6 GF/s
4 MiB* eDRAM

(compare this with a 1988 Cray YMP/8 at 2.7 GF/s)

Compute Card
I/O Card

FRU (field replaceable unit)
25mmx32mm
2 nodes (4 CPUs)
(2x1x1)
2.8/5.6 GF/s
256/512 MiB* DDR

Node Card

16 compute cards
0-2 I/O cards
32 nodes
(64 CPUs)
(4x4x2)
90/180 GF/s
8 GiB* DDR

Cabinet

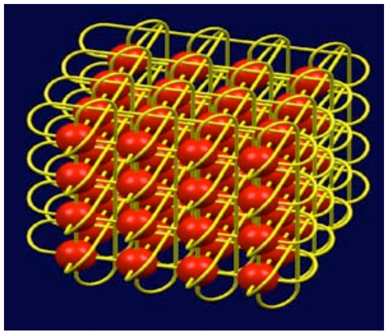
2 midplanes
1024 nodes
(2,048 CPUs)
(8x8x16)
2.9/5.7 TF/s
256 GiB* DDR
15-20 kW

System

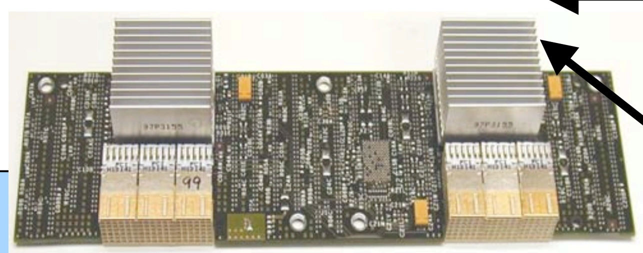
64 cabinets
65,536 nodes
(131,072 CPUs)
(32x32x64)
180/360 TF/s
16 TiB*
1.2 MW
2,500 sq.ft.
MTBF 6.16 Days

* http://physics.nist.gov/cuu/Units/binary.html

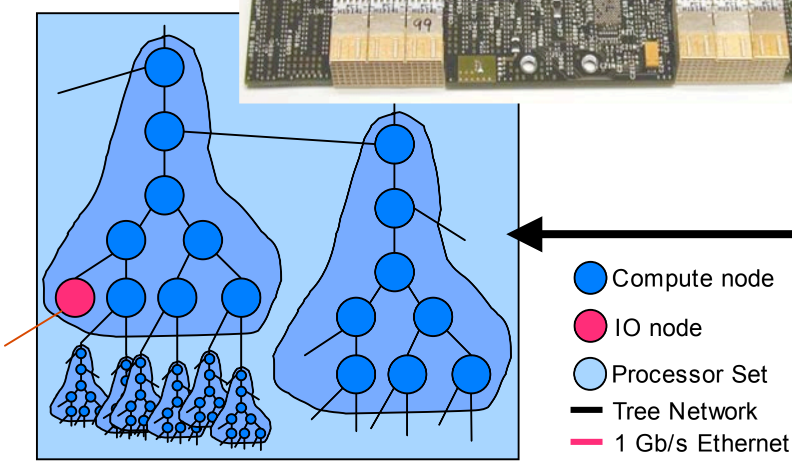
Architectural features promote efficiency and scaling for important applications



- **Multiple complementary interconnects support diverse application scaling requirements**
  - 3D torus with bi-directional nearest-neighbor links
  - 2.1 GB/s combining tree for fast global reductions
  - Low-latency global barrier network



- **High reliability expected from high level of integration using system-on-a-chip technology**
- **Architectural enhancements improve single node performance**



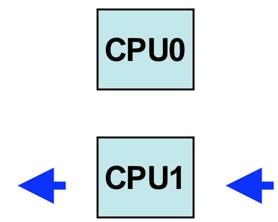
- **Software architected with very powerful "divide and conquer" technique for software scale-up**

The BG/L project is a focused effort to enable important science and to lead the way to cost-effective petaFLOP/s computing

Two ways for apps to use hardware

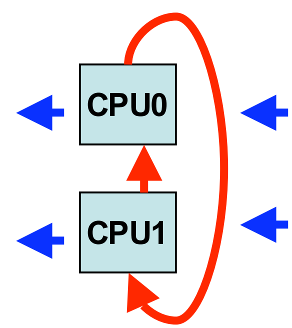
Mode 1 (Co-processor mode - CPM):

- CPU0 does all the computations
- CPU1 does the communications
- Communication overlap with computation
- Peak comp perf is 5.6/2 = 2.8 GFlops

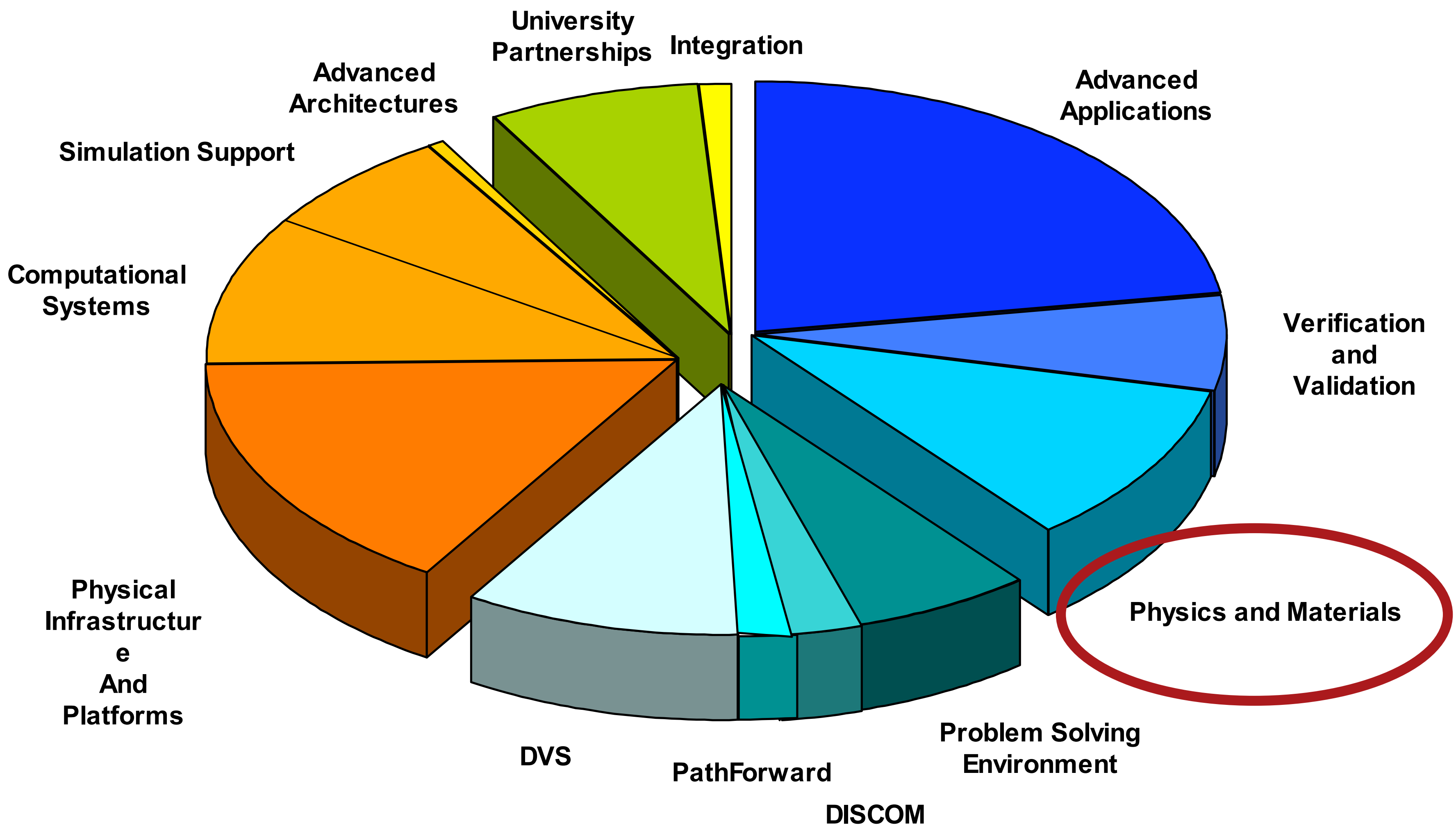


Mode 2 (Virtual node mode - VNM):

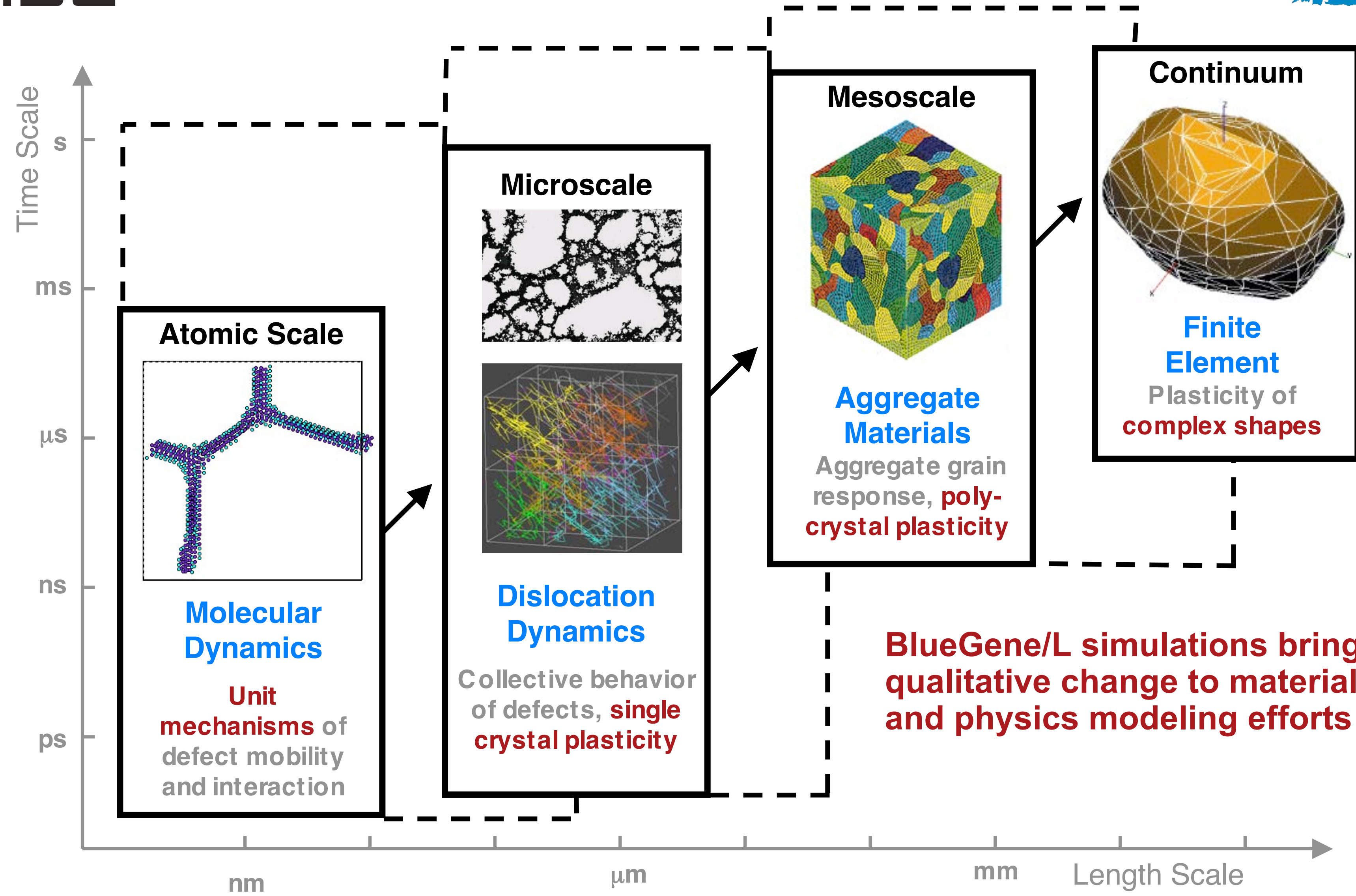
- CPU0, CPU1 independent "virtual tasks"
- Each does own computation and communication
- The two CPU's talk via memory buffers
- Computation and communication cannot overlap
- Peak compute performance is 5.6 GFlops



Physics and Materials is initial ASC focus for the early BlueGene/L apps



BlueGene/L will allow overlapping evaluation of models for first time



Predicting the mechanical strength of material from first principles is difficult

- *In fact, it has remained a grand challenge for computational materials science for several decades…*

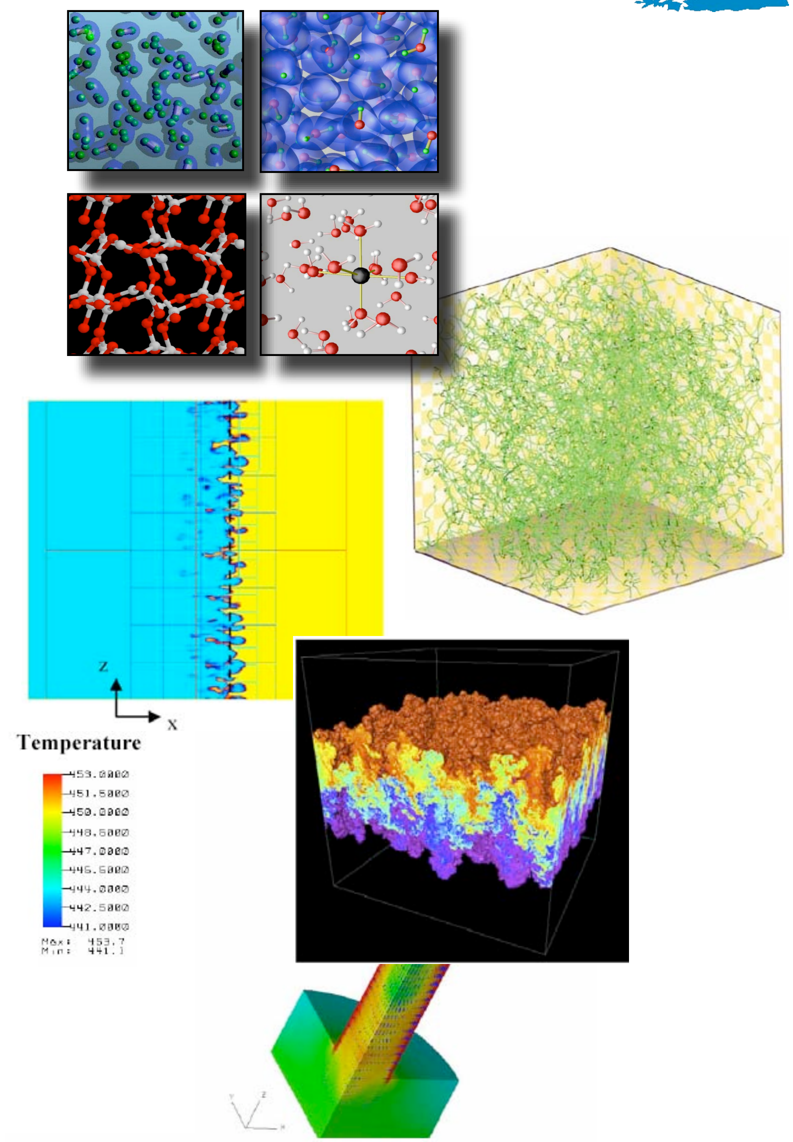
- **At a fundamental scale, we need to consider the atomistic structure of a "dislocation"** and the atomistic mechanisms of dislocation motion.

- **At higher scale, we need to consider interaction of many dislocations that form complex patterns on a micron scale** to understand the plastic strength of a single crystal.

- **Single-crystal strength is used to model a poly-crystal of many crystal grains,** which in turn supports yet higher-scale finite-element models of complex-shape object deformation.

Criteria for "First-Wave" Applications

- **First-wave target ASC applications**
  - **Efforts identified to be ready for programmatic science runs with early machine availability, and ongoing assessment of code suitability**
- **Assessment criteria for early apps**
  - **Importance to the ASC program**
  - **Enthusiasm within the code group**
  - **Potential for good code scaling *(i.e., simpler architectural needs)***
- **Second-wave target ASC apps are also being identified using similar criteria to those above**



Also examining many other science applications to run on BlueGene/L

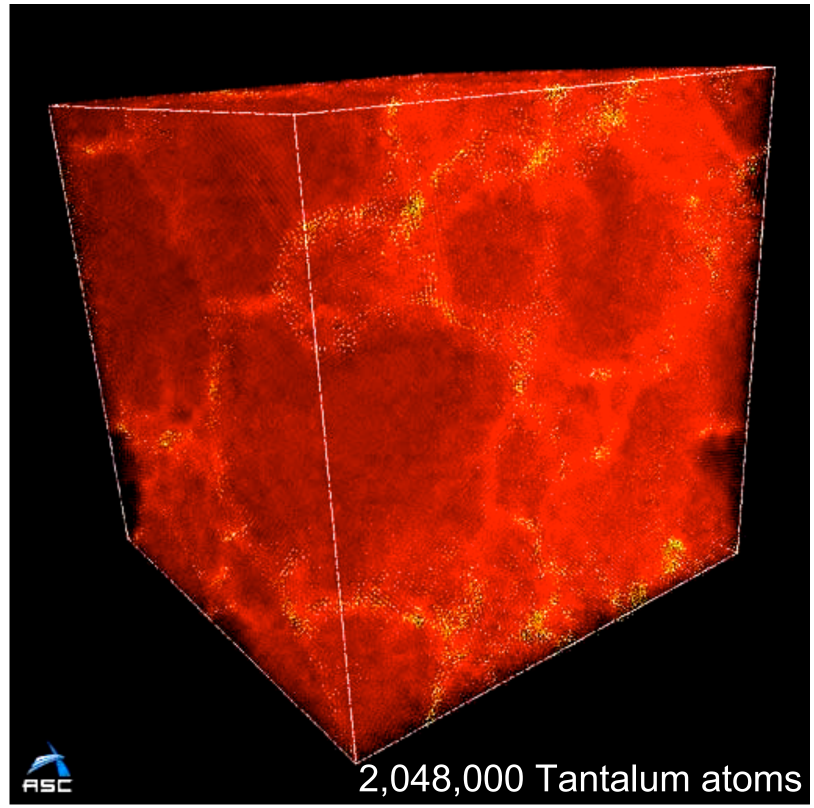
- **ALPS** – Predictive modeling of laser plasma interaction
- **AMRh** – High Reynolds-number fluid flows
- **BOUT** – MFE boundary-layer turbulence
- **DJEHUTY** – 3D modeling of stars
- **DYNA3D** – Structural mechanics
- **EMSOLVE** – Electromagnetic coupling with structures
- **FMC** – Solves Schroedinger Equation for a many-fermion system
- **GFMD** – Greens-function molecular dynamics
- **HYDRA** – Implicit radiation diffusion solver, multi-mode instabilities
- **IRS** – Implicit radiation solver
- **ParaDyn** - LLNL parallel engineering code based on DYNA3D
- **PF3D/Z3** – Predictive modeling of laser plasma interactions
- **ROLEX** – Detailed-accounting opacity
- **SAGE** – LANL widely used adaptive-grid Eulerian hydrodynamics

Classical MD – ddcMD: Rapid resolidification in tantalum

• Scalable, general purpose code for performing classical molecular dynamics (MD) simulations using highly accurate MGPT potentials

• MGPT semi-empirical potentials, based on a rigorous expansion of many body terms in the total energy, are needed in to quantitatively investigate dynamic behavior of transitions metals and actinides under extreme conditions

64K and 256K atom simulations on 2K nodes are order of magnitude larger than previously attempted; based on 2M atom simulation on 16K nodes, *close to perfect* scaling expected for full machine ("very impressive machine" says PI…)



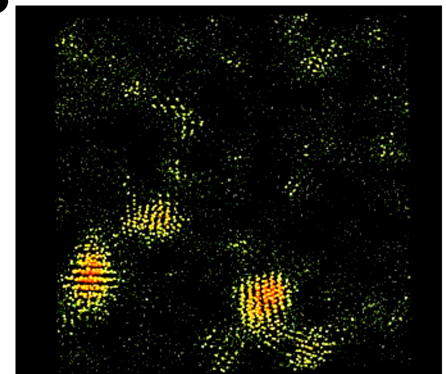
Visualization of important new scientific findings already achieved on BG/L: Molten Ta at 5000K demonstrates solidification during isothermal compression to 250 GPa

Excellent scaling of ddcMD on BG/L supports greater understanding of resolidification process

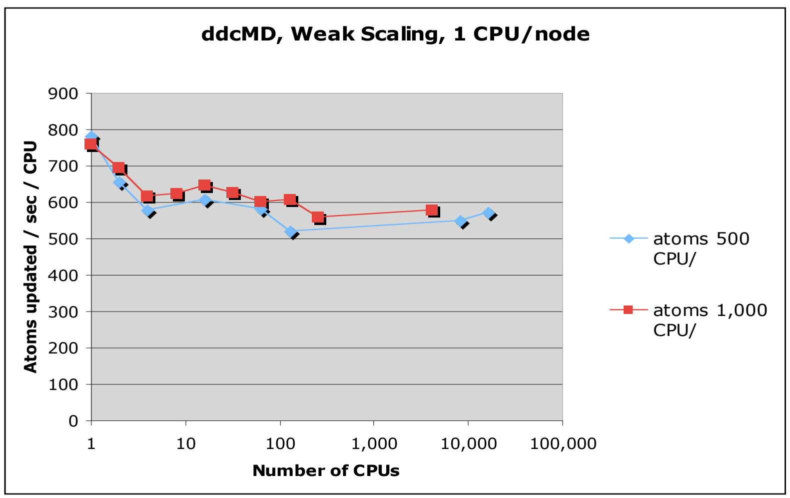
• Nucleation of solid is initiated at multiple independent sites throughout each sample cell

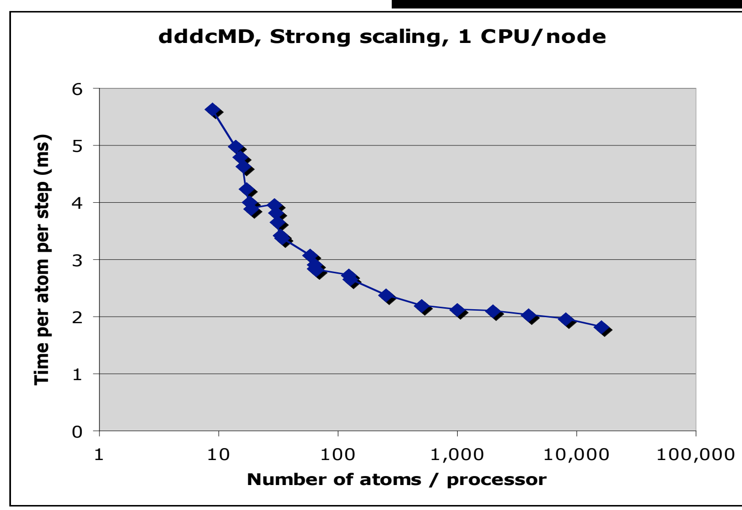
• Growth of solid grains initiates independently, but soon leads to grain boundaries which span the simulation cell: size of cell is now influencing continued growth

•2,048,000 simulation recently performed indicates formation of many more grains



64K Tantalum atoms





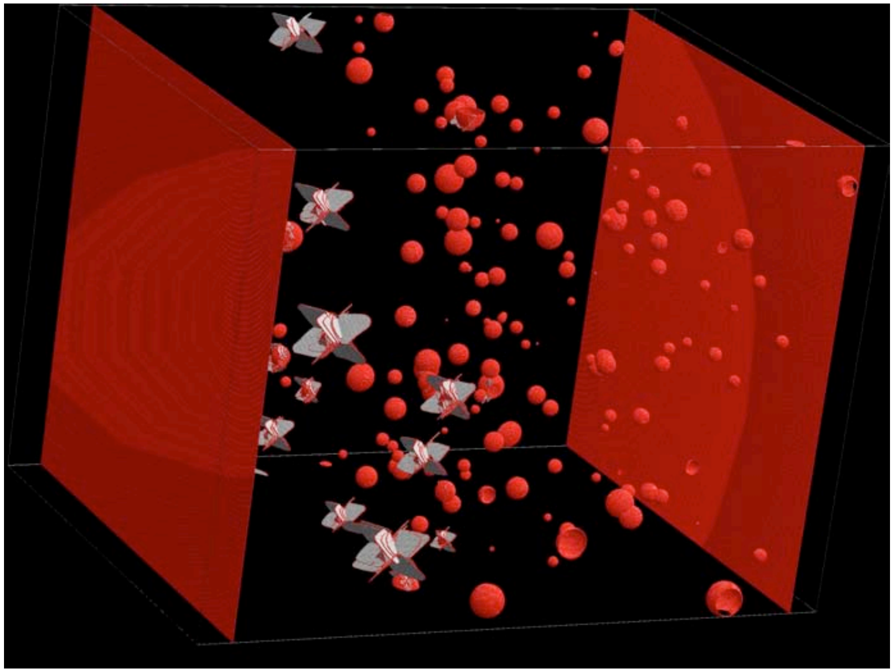
ddcMD is already using 32K* CPUs of BG/L for unprecedented multi-million atom MGPT simulations

* Virtual node mode gives 1.9x performance out to 1K processors

Classical MD - SPaSM

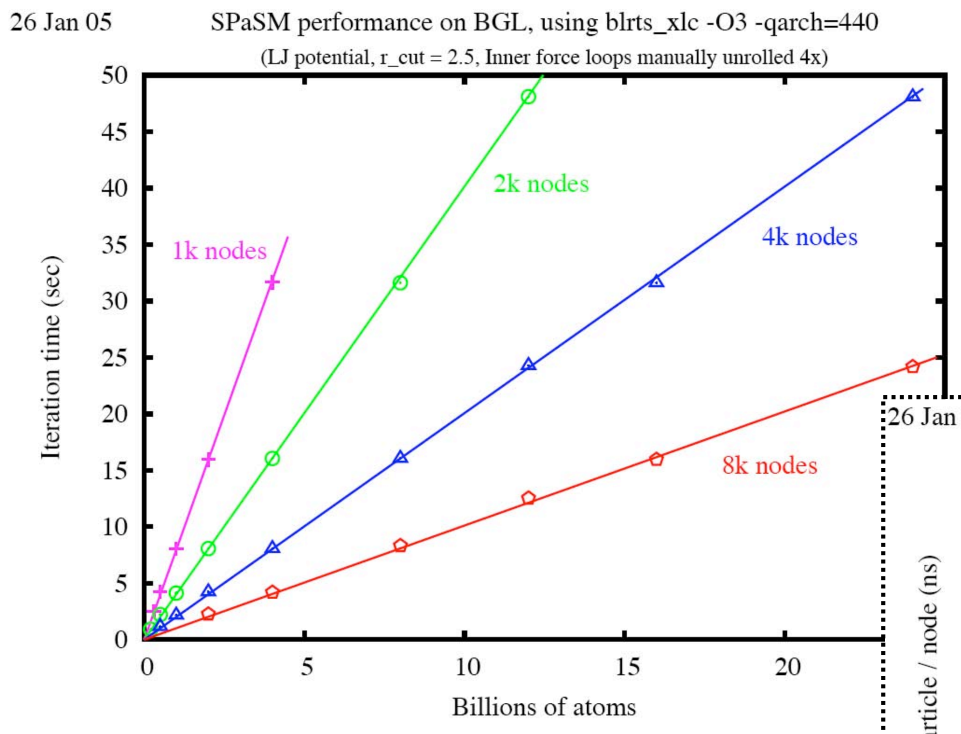
- A high performance (1993 and 1998 Gordon Bell prizes) code for **S**calable **Pa**rallel **S**hort-range **M**olecular dynamics simulations

- A variety of finite-range empirical potentials are implemented, including EAM and MEAM for metals, Stillinger-Weber Si/Ge, and a reactive empirical bond-order (REBO) potential for detonation studies.

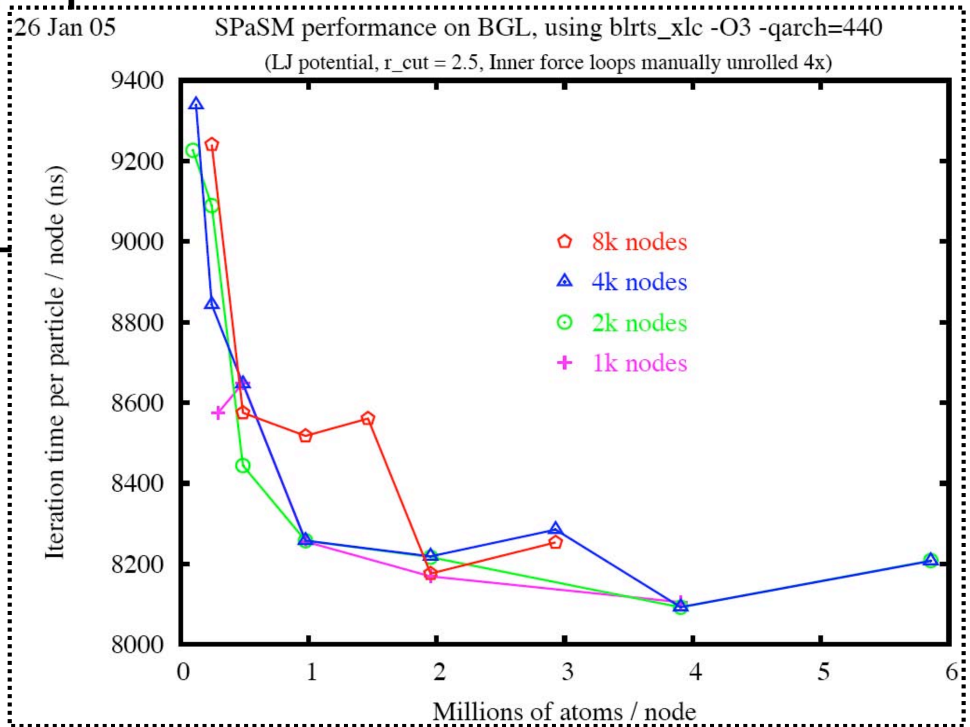


SPaSM has exhibited excellent scaling for up to 100 billion atoms on 16,384 nodes, and an initial production run on 8k nodes simulated the shock loading of a 2.1 billion atom EAM copper crystal with 0.41% (by volume) voids. BG/L will enable the exploration of an entirely new class of (previously intractable) problems such as this.

SPaSM performance on BG/L with 100,000 — 6,000,000 atoms per node



- Excellent scaling is seen for >1 million atoms per node
- This will extend to smaller sizes for more complex potentials, which typically have much higher CPU/communication ratios



GRASP: Scalable Molecular Dynamics Code for Reactive Force Fields (SNL)

• A scalable, general purpose code for performing classical molecular dynamics (MD) simulations

• Supports a wide range of different force fields: twobody, threebody, Tersoff, EAM, ReaxFF, electrostatics, charge equilibration

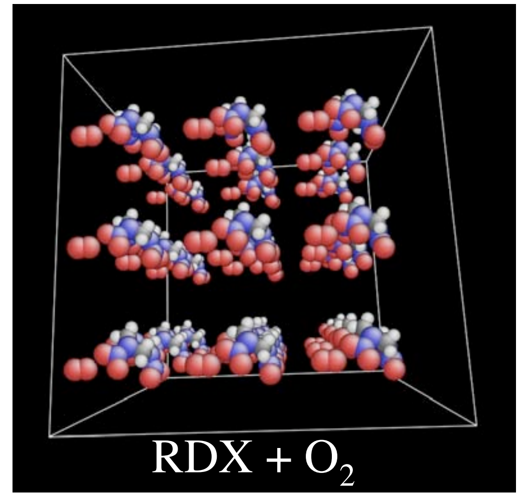
•Applications include: Radiation Damage, Materials Interfaces, Explosives, MEMS

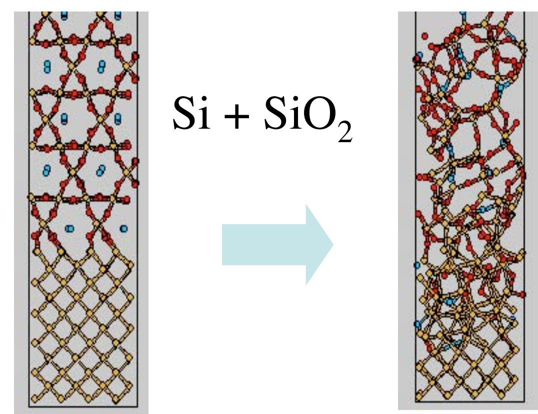
•Standard version ported to BGL without difficulties

•Development version combining C++ and Fortran implements ReaxFF force field. Used BGL to test the code. Several software bugs detected and fixed.

•Absolute speed for ReaxFF close to 3 GHz Pentium cluster.

•Stillinger-Weber silicon benchmark scales well
  – 70% efficiency at 122 atoms/CPU on 16k processors.
  – 35% efficiency at only 4 atoms/CPU on 4k processors.





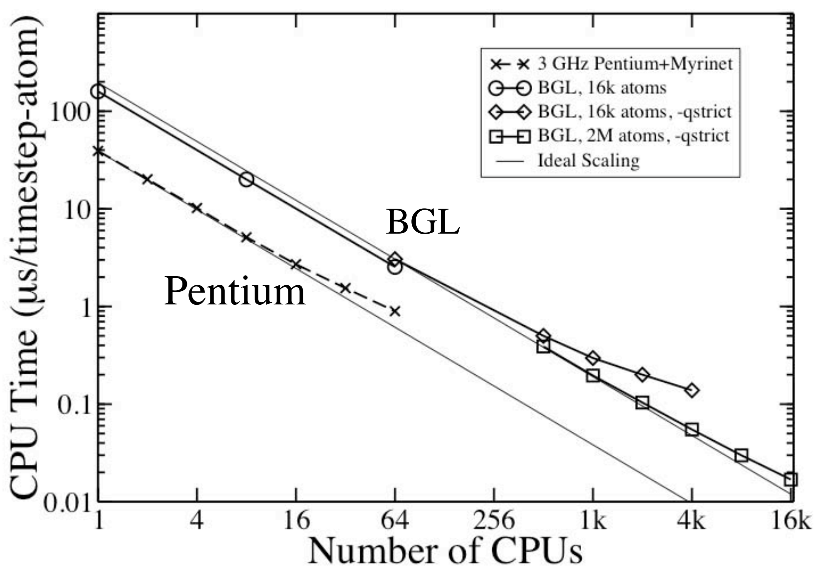
GRASP performance on BG/L

and comparison to HP cluster (3 GHz Pentium+Myrinet

α-Silicon crystal

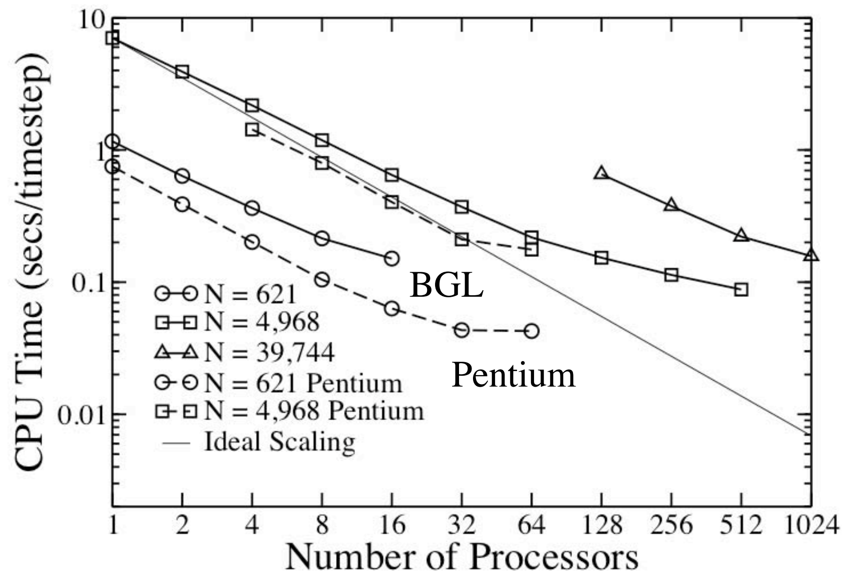
Rhombohedral periodic cell

Stillinger-Weber force field

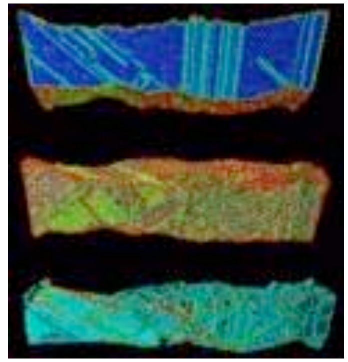


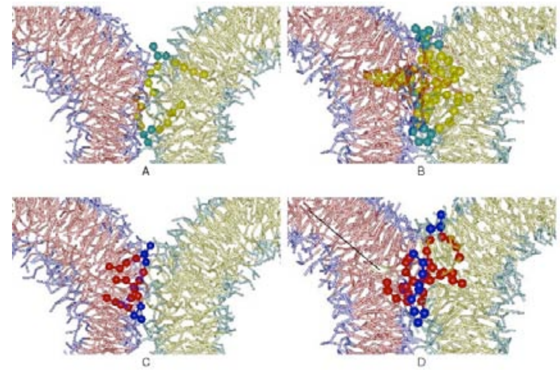
RDX Explosive with Oxygen

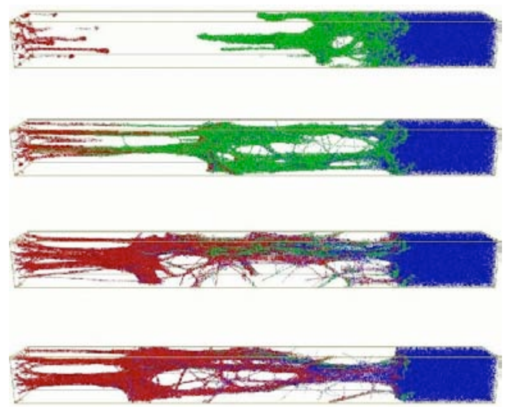
ReaxFF force field with charge equilibration

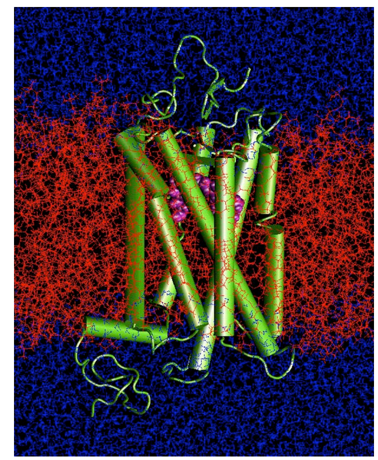


LAMMPS Classical MD (SNL)









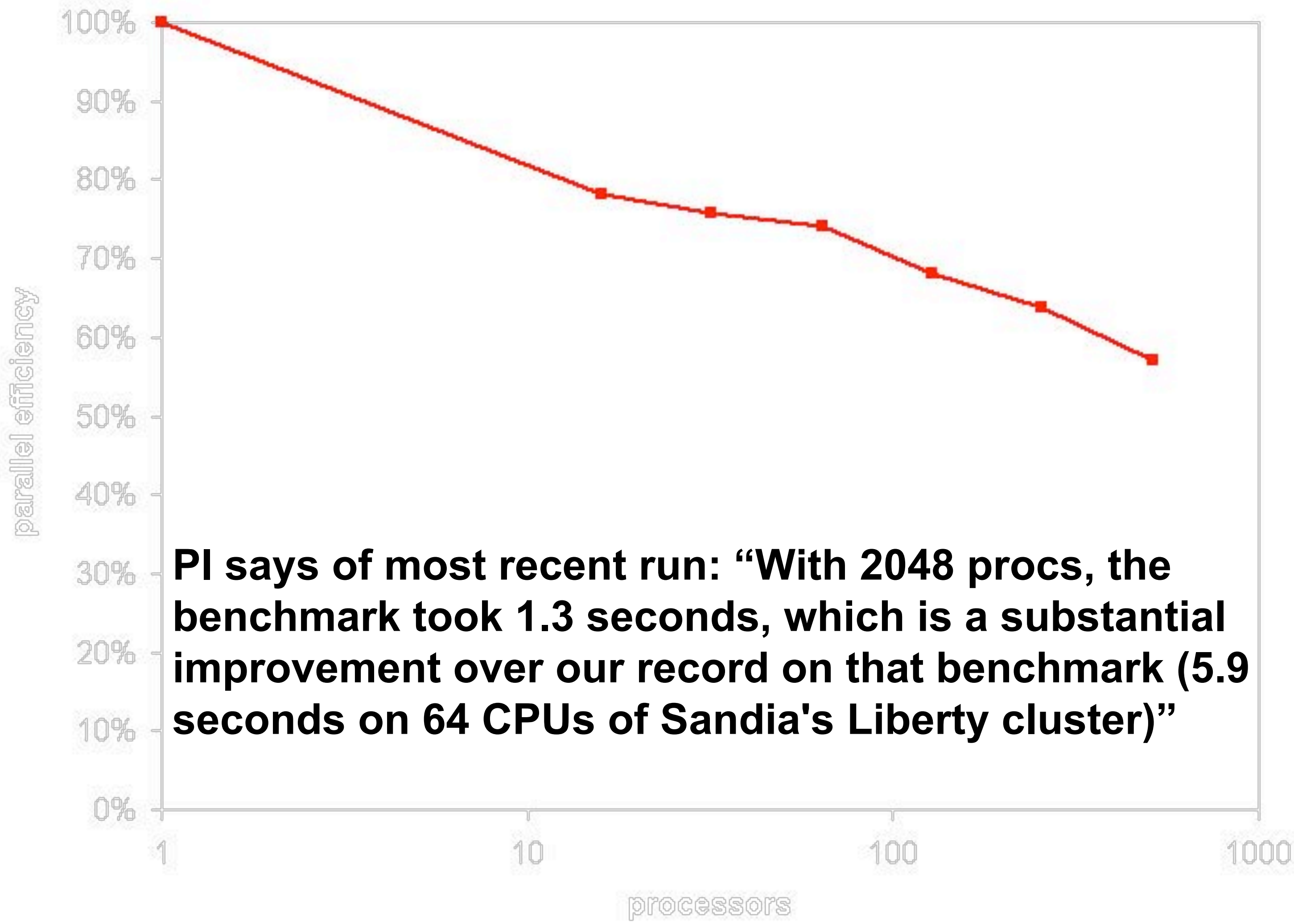
- LAMMPS = Large-scale Atomic/ Molecular Massively Parallel Simulator

- LAMMPS is a classical molecular dynamics code that models an ensemble of particles in a liquid, solid, or gaseous state. It can model atomic, polymeric, biological, metallic, or granular systems using a variety of force fields and boundary conditions.

- On parallel machines, LAMMPS uses spatial-decomposition techniques to partition the simulation domain into small 3d sub-domains, one of which is assigned to each processor.

LAMMPS has been tested on up to 512 BG/L processors so far, and shown good scaling on a fixed-size (32,000 atoms) problem (strong scaling).

LAMMPS strong scaling on BG/L with 32,000 atom fixed size problem



PI says of most recent run: “With 2048 procs, the benchmark took 1.3 seconds, which is a substantial improvement over our record on that benchmark (5.9 seconds on 64 CPUs of Sandia's Liberty cluster)”

Classical MD - MDCASK

- MDCASK simulates the motion of large collections of individual atoms using the classical laws of Newtonian mechanics and electrostatics.
- Capable of using a wide variety of inter-atomic potentials allowing simulation of metals, semiconductors, insulators, and glasses

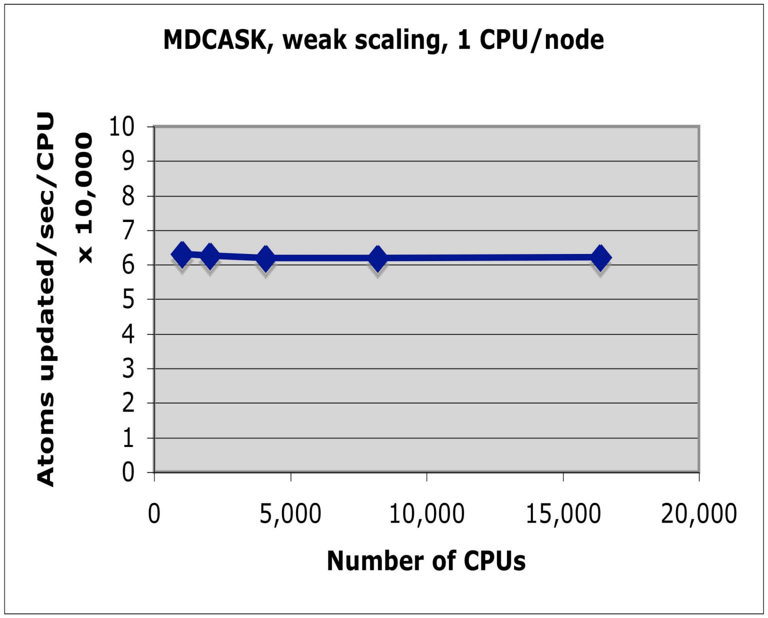
• Weak scaling (with a constant 250,000 atoms per processor) was tested up to 16,384 processors with excellent results.

• Virtual node mode yields a factor of 1.78 speedup.

• To simulate 1 ns with 10^{10} atoms requires ~ 8 days on the full-sized BG/L.

• Strong scaling tests perform well down to ~2,000 atoms / node.

MDCASK is ready to apply the full power of BGL to multi-billion atom simulations.



First-Principles MD - Qbox

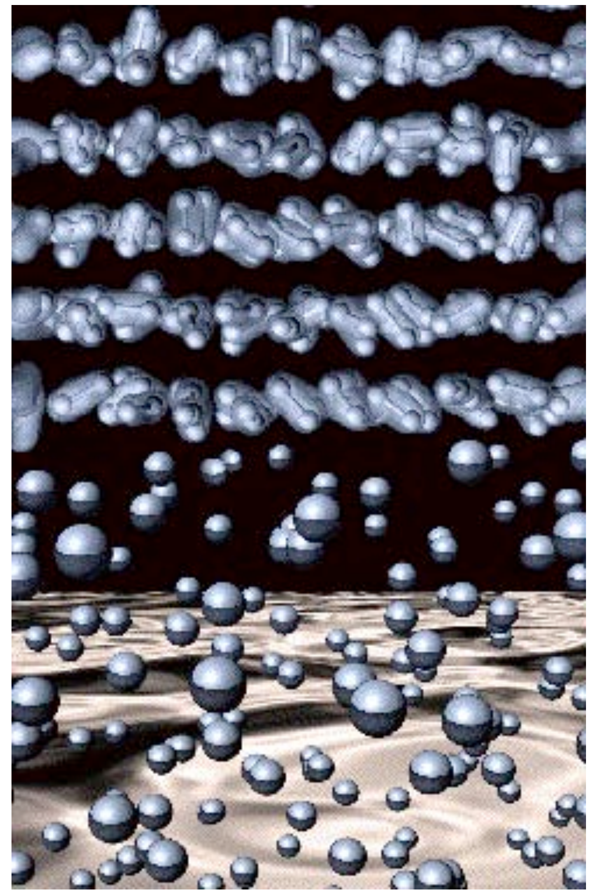
•Qbox is a C++/MPI implementation of the plane-wave, pseudopotential, ab initio molecular dynamics method within Density Functional Theory (DFT). It is developed at LLNL.

•Massively parallel C++ / MPI implementation with specialized 3D FFTs

•Routinely used at LLNL for simulations of condensed matter subjected to extreme such as high pressure and high temperature, as well as in nanotechnology and biochemistry applications.

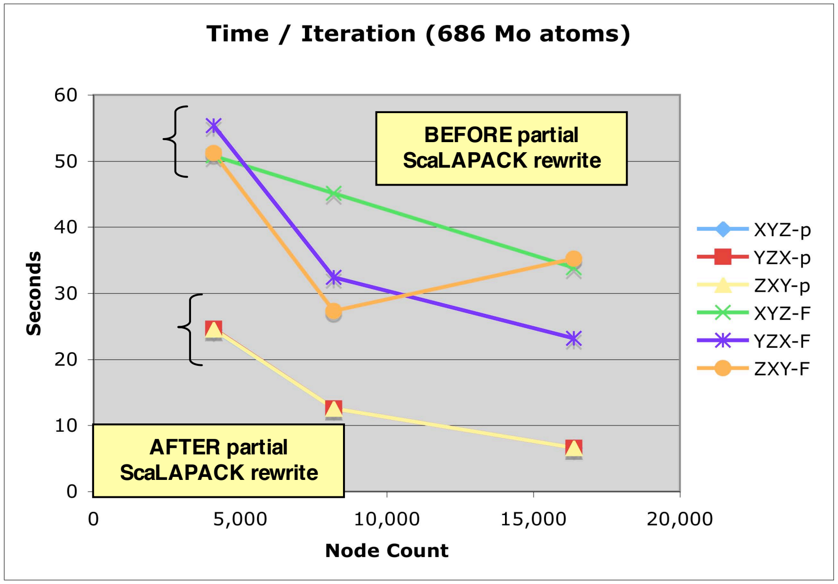
• 686-atom Mo solid and other heavy metal simulations are under way

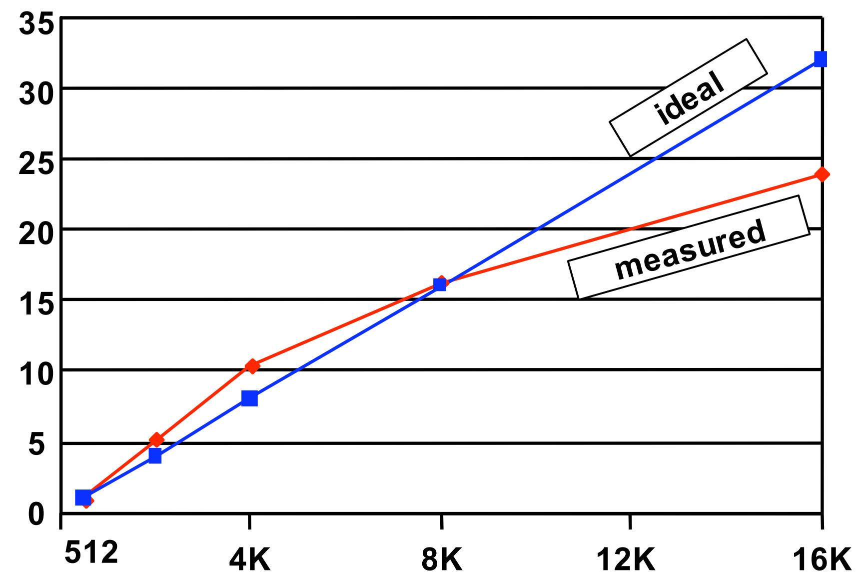
•Scalability tests on BG/L show that Qbox can achieve a 3x speedup when solving a given problem on 16384 nodes instead of 4096 nodes. This represents a 75% parallel efficiency. Further optimizations will provide even greater efficiency.



This figure (generated with GP, pre-cursor of Qbox) was recently used as the cover of the October 7, 2004 issue of the journal Nature.

Qbox: (strong) scaling on BG/L
Solid Molybdenum simulation

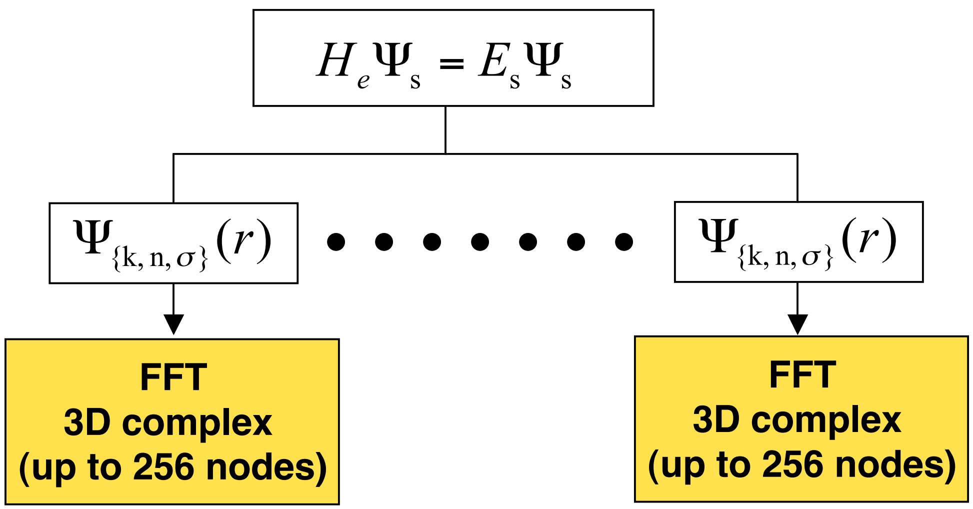




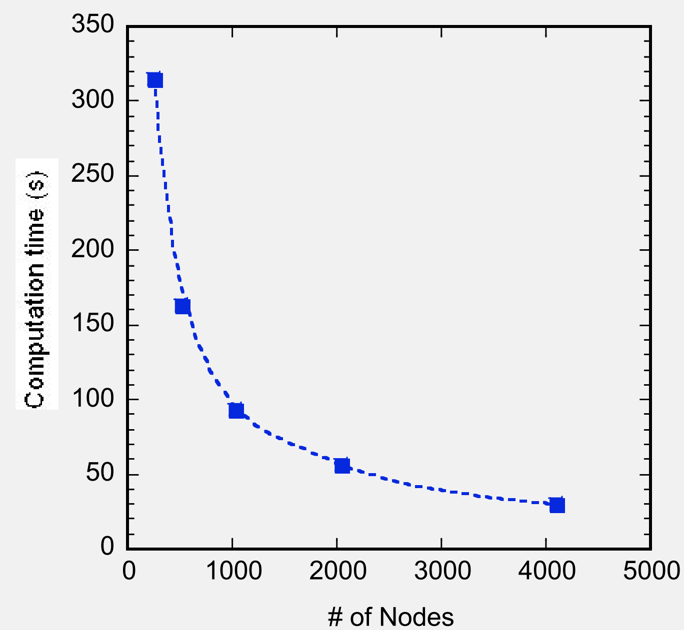
- Some lessons learned:
  - Node mapping is critical, can result in a 2x speedup
  - Using two CPUs / node yields ~1.5x speedup
  - Mixed “AIX/Linux” development environment, w/evolving compilers and nascent MPICH-2 BG/L device, has proved challenging
  - 16k task algorithm scaling frequently requires modifications: Rewrote some ScaLAPACK functions to improve scaling above 4k nodes
- Current efforts target generating efficient node mappings, optimization of linear algebra operations and parallel I/O

First-Principles MD - FEQMD

- **FEQMD has required a complete "inversion" of its parallelization strategy.**
- **Data arrangement: {X, Y, Z} in real space, {Z, Y, X} in Fourier space reduces the number of transpose operations.**
- **Further scaling and optimization work is underway.**



Runs on BG/L for 64 atoms, one self-consistent calculation without I/O



BGL is ~25x the power of ASCI Q, where we currently simulate 128 atoms

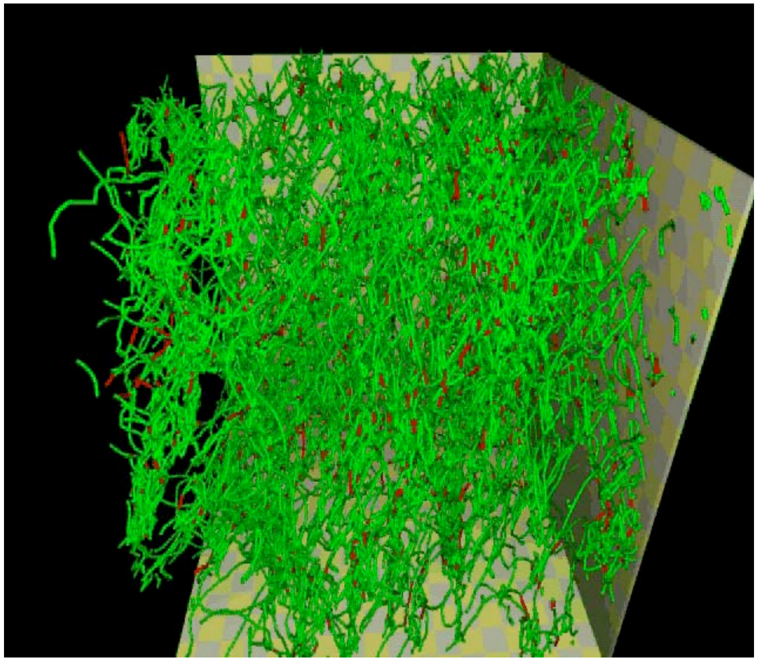
Dislocation Dynamics - ParaDiS
(Parallel Dislocation Simulator)

• New LLNL code for direct computation of plastic strength of materials

• Tracks simultaneous motion of millions of dislocation lines

• Promises to close the computational performance gap that prevents scientists from understanding the fundamental nature of material strengthening (or hardening)

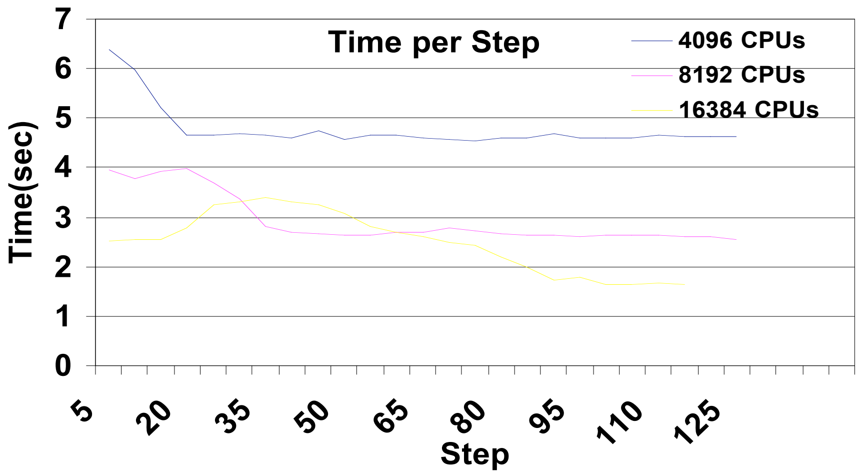
ParaDiS has run on 16,384 nodes of BG/L, and is currently investigating scaling and dynamic load balancing issues to achieve higher efficiencies.

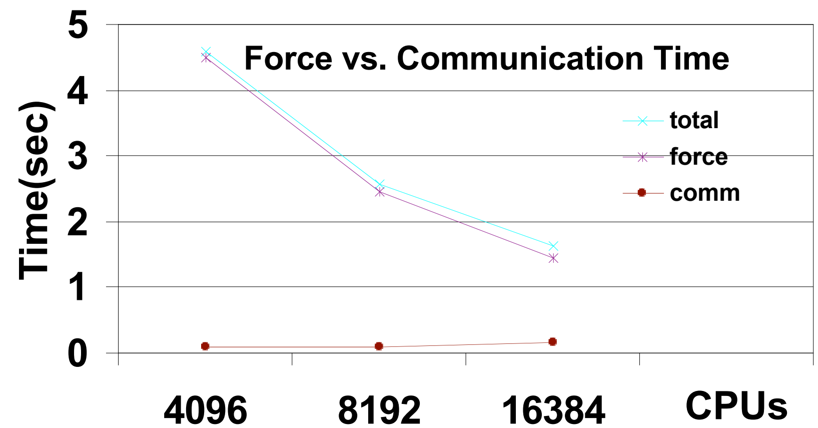


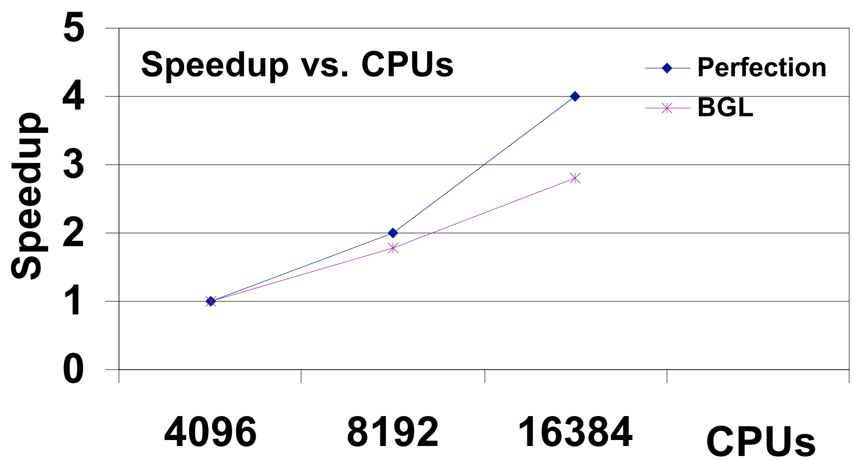
Killer applications:

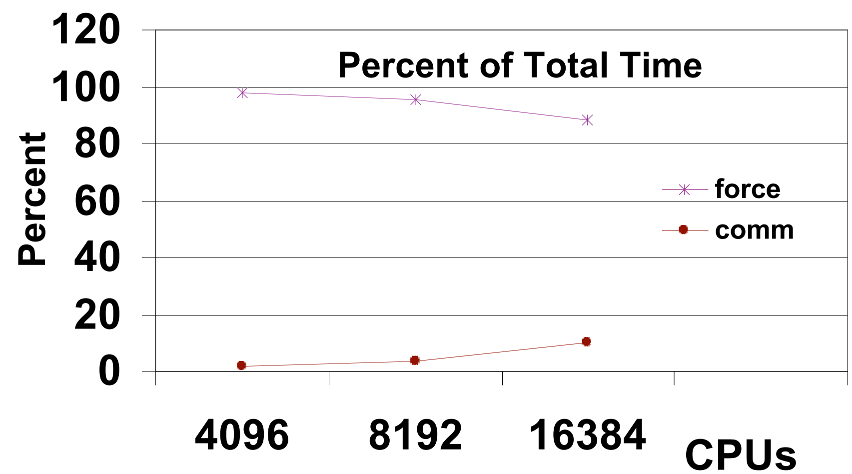
- full simulation of poly-crystal solidification from melt
- alloy microstructure evolution during plastic deformation
- science of ultra-fast polymer crystallization

Dynamic load balancing key to scaling ParaDis to large node counts



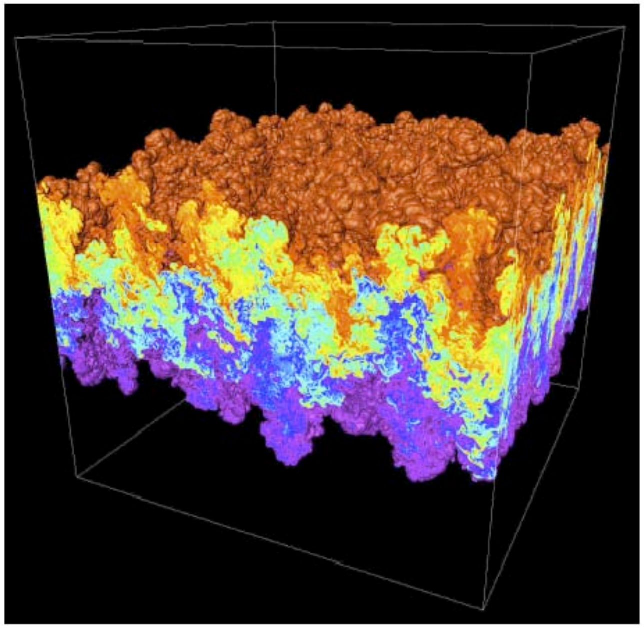






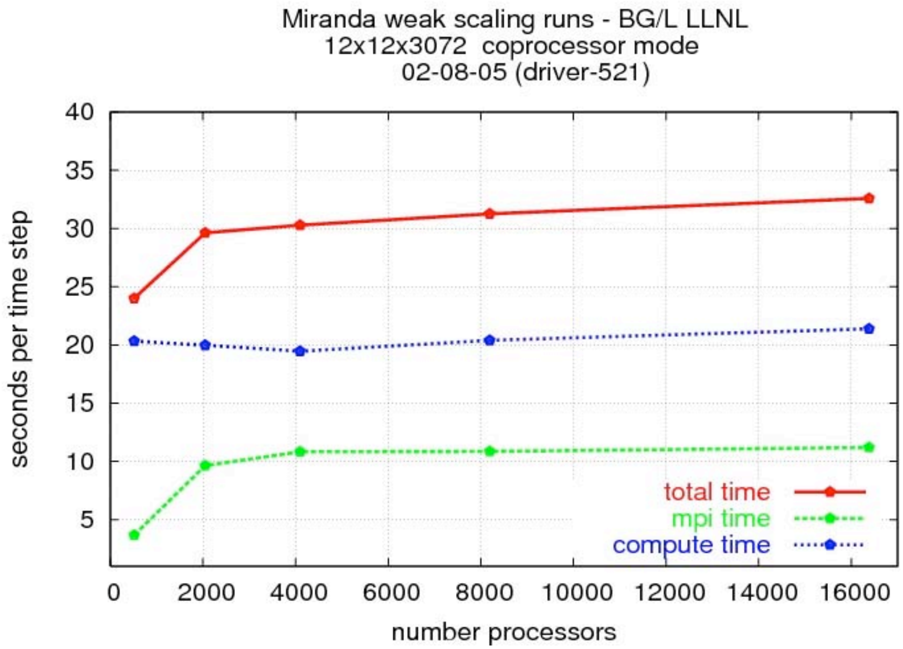
Instability and Turbulence - Miranda

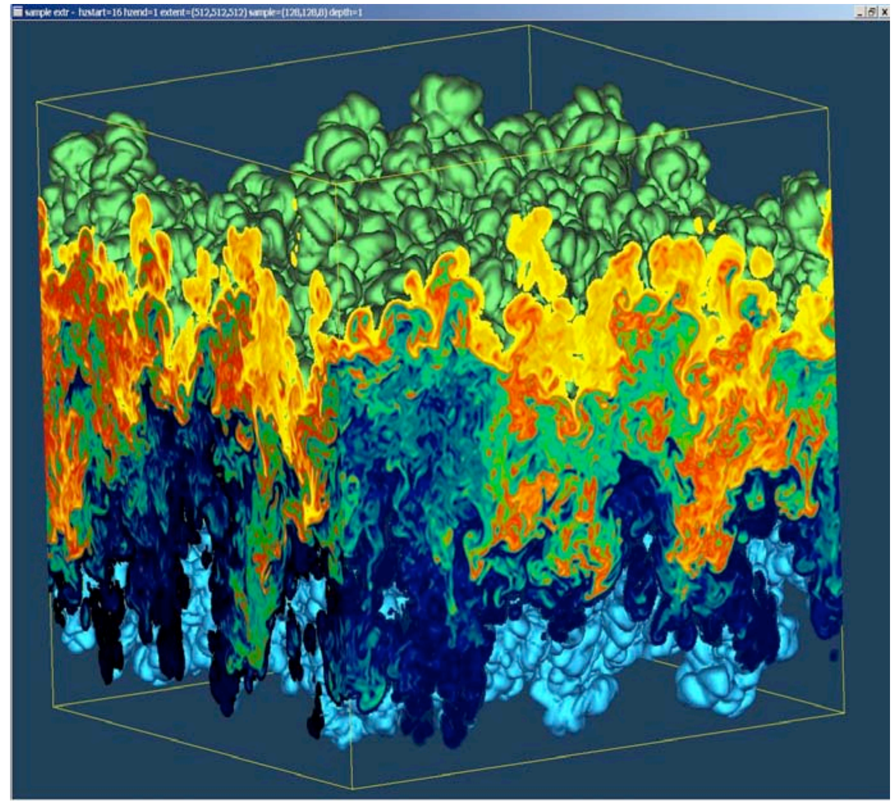
- High order hydrodynamics code for computing fluid instabilities and turbulent mix
- Employs FFTs and band-diagonal matrix solvers to compute spectrally-accurate derivatives, combined with high-order integration methods for time advancement
- Contains solvers for both compressible and incompressible flows
- Has been used primarily for studying Rayleigh-Taylor (R-T) and Richtmyer-Meshkov (R-M) instabilities, which occur in supernovae and Inertial Confinement Fusion (ICF)



Miranda has successfully run on 16,384 nodes on BG/L and also on 32,768 processors in “virtual node” mode. BG/L enables wide range of scales in space and time necessary to represent turbulent flows of interest. Good time-to-solution improvement from MCR to BG/L .

Miranda Weak Scaling on BG/L



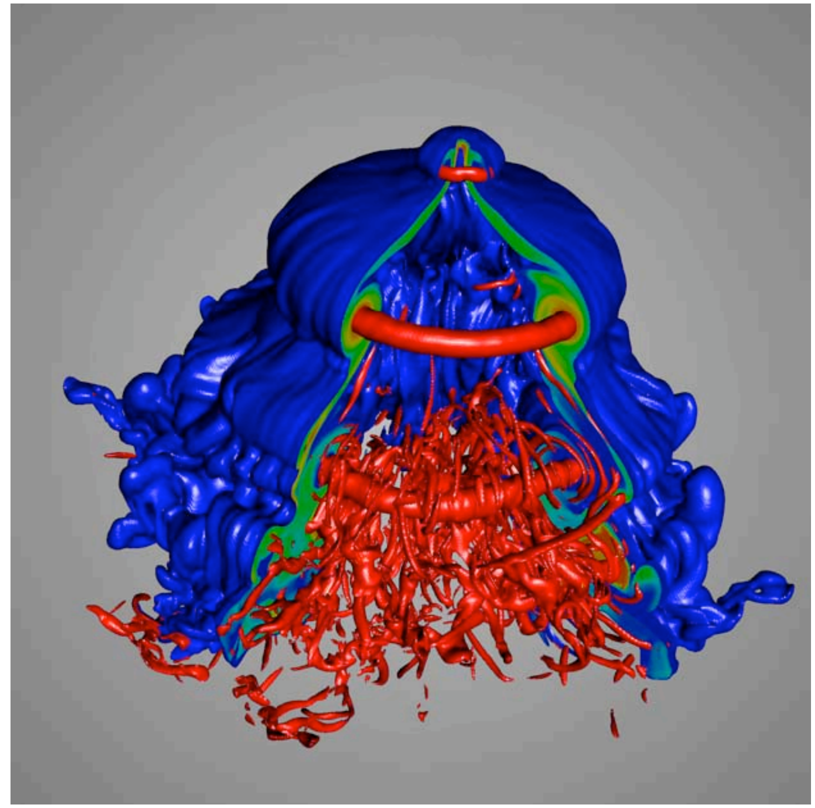


Instability and Turbulence - Raptor

• Multi-physics Eulerian Adaptive Mesh Refinement (AMR) code used for applications at LLNL including astrophysics, Inertial Confinement Fusion (ICF) and shock-driven instabilities and turbulence

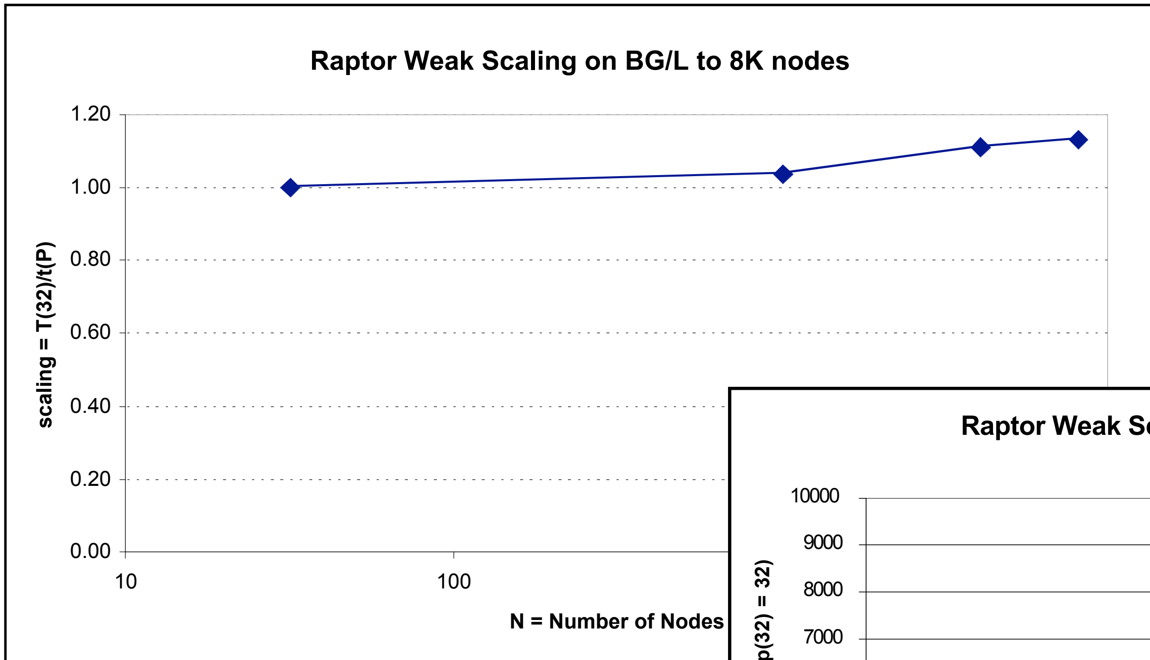
• Can be used to simulate purely fluid dynamics systems and more complex physical systems where the fluids are coupled to the radiation field, such as in ICF or astrophysics

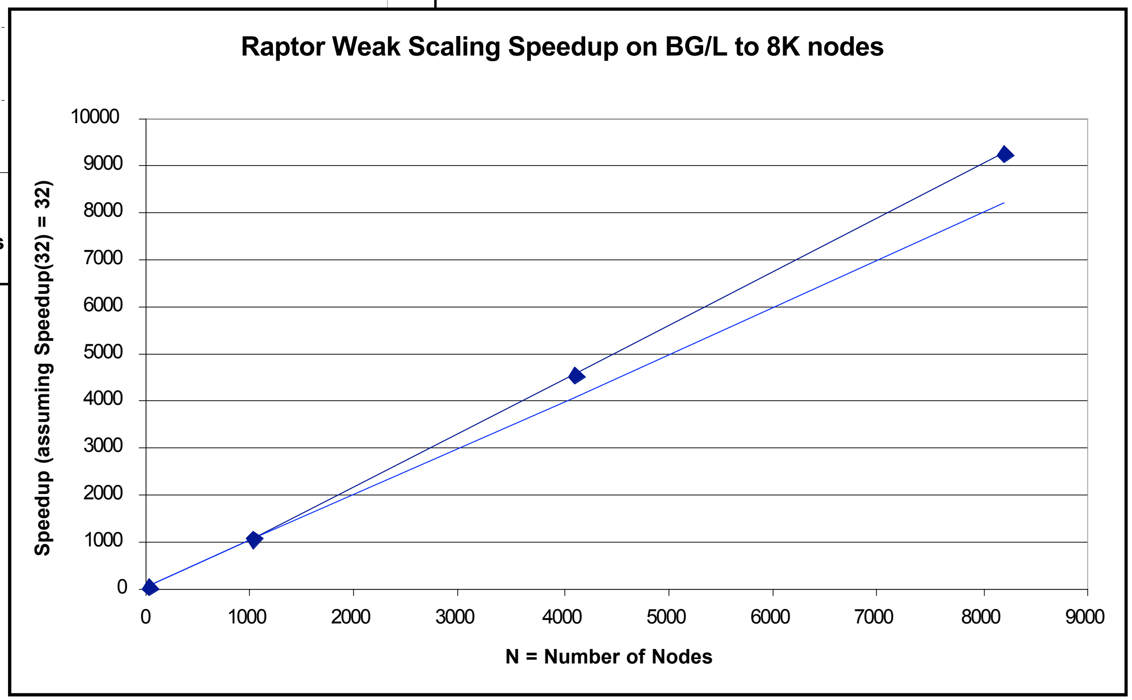
Simulations at full scale on BG/L will offer the computational power to gain an order of magnitude more resolution in simulations of three-dimensional shock-driven systems.



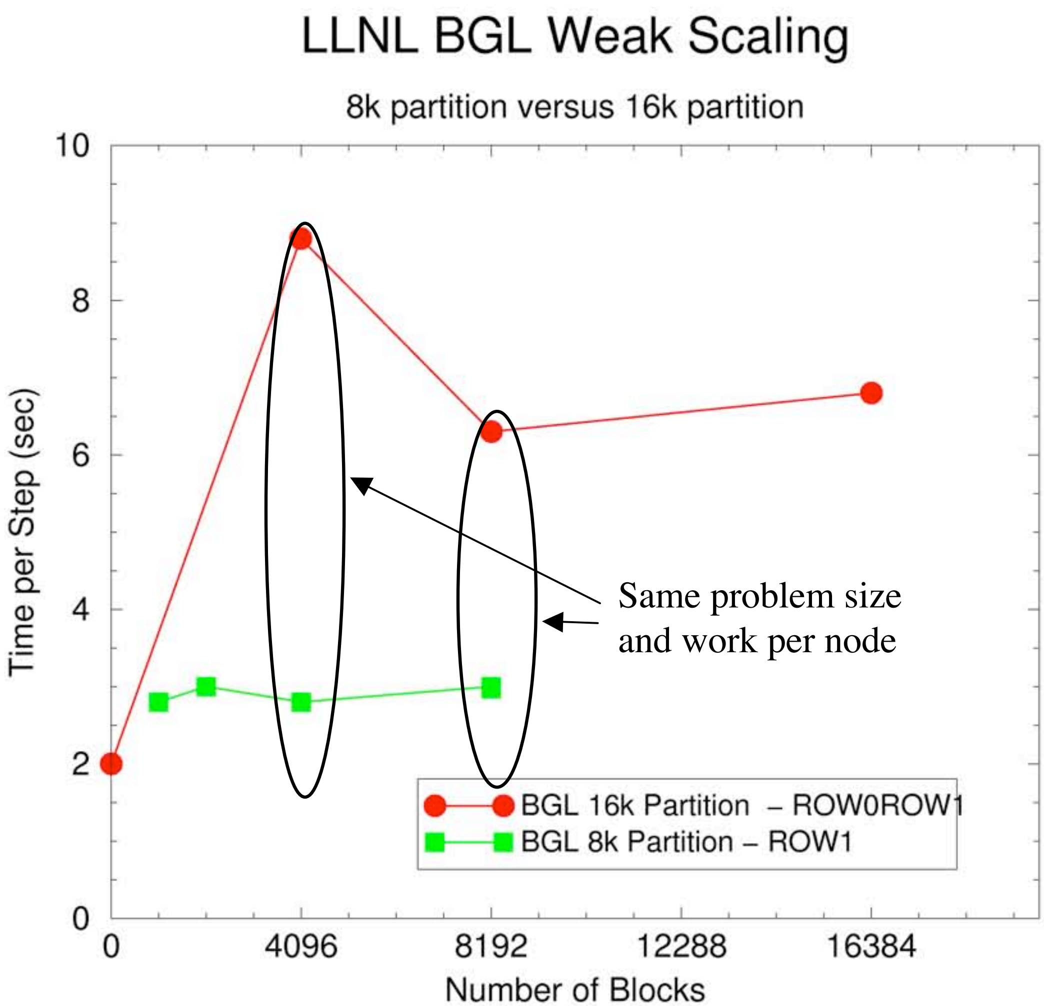
Dense spherical argon bubble, initially contained in a thin spherical soap film, suspended in nitrogen, subjected to a strong planar shock wave about 509 microseconds after shock-bubble interaction. Blue represents the argon iso-surface, red indicates vorticity magnitude, and the film material volume fraction is plotted on the cross-sectional cut planes.

Good scaling results seen for Raptor (note better than linear weak scaling)





Current Raptor Scaling Efforts Focus on 16K Performance Anomaly



• Current efforts aimed at understanding differences between 8k and 16k performance.

• A communications kernel that simulations the point-to-point message "storms" in Raptor has been written and tested.

Planned Multi-Physics Runs - ALE3D (not yet running, export control issue)

Explosives Applications

Using the capability of ALE3D and the resources of BG/L will simulate processes by which chemical explosives detonate (from intended or accidental stimuli) to develop safer explosives

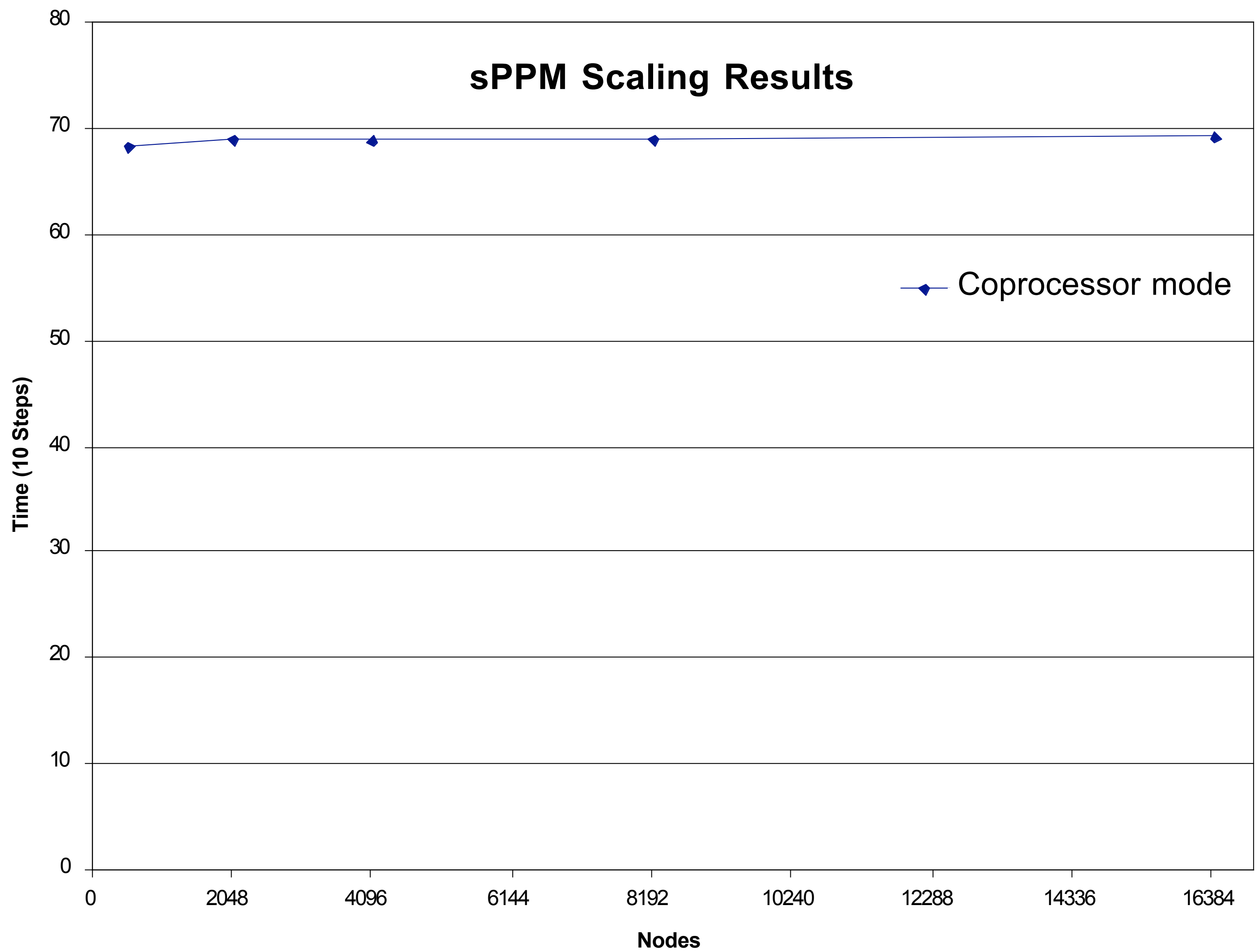
Poly-crystal Plasticity

Fragmenting cylinder calculations in ALE3D will be the first full-scale simulations using material behavior determined directly from a poly-crystal plasticity model

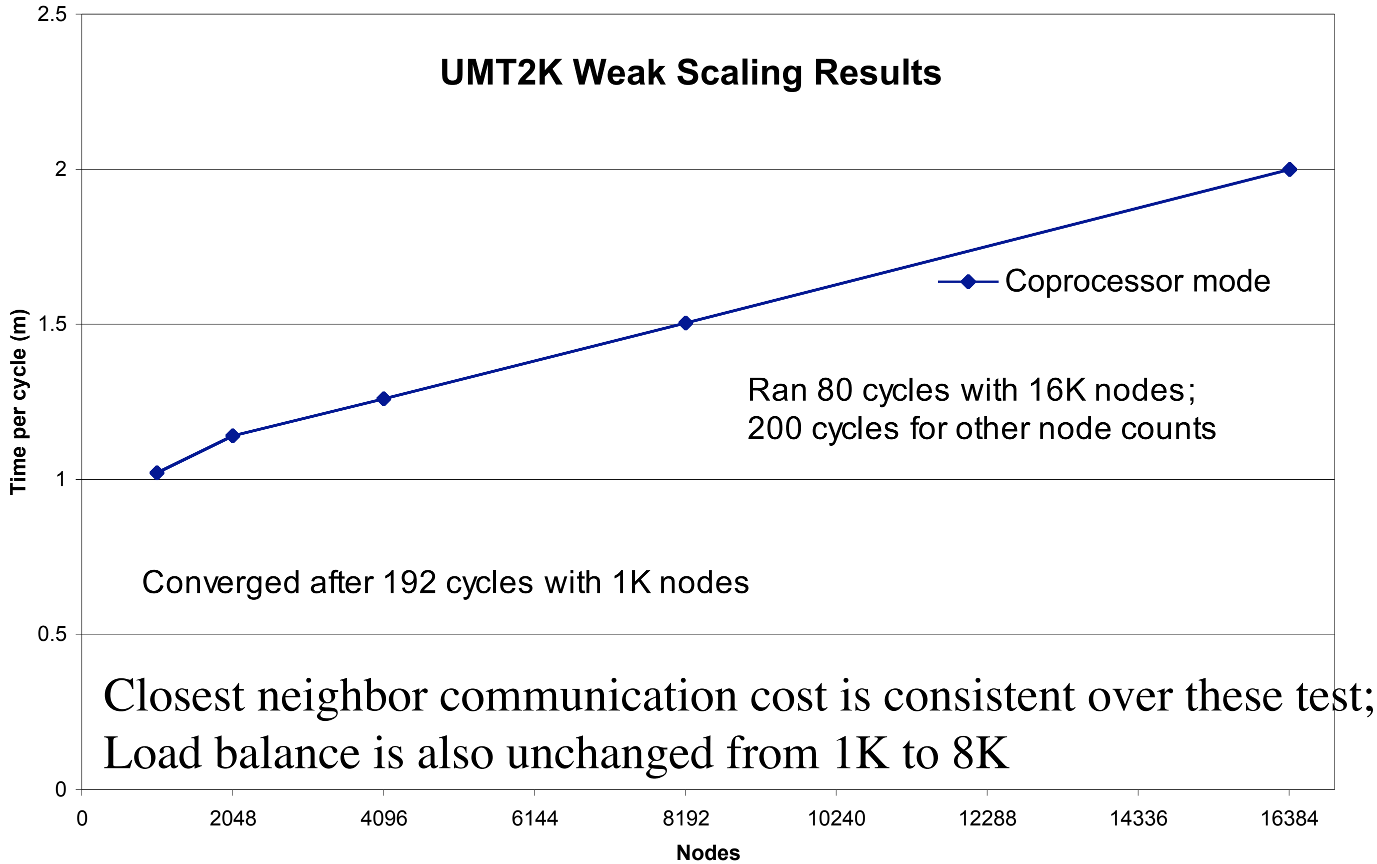
Military Thermal Safety

With ALE3D and BG/L, LLNL will be the first research lab to simulate cookoff in the various phases of high-explosive heating and phase change

sPPM demonstrates near perfect weak scaling to 16,384 nodes



Recent UMT2K runs demonstrate good performance up to 8192 nodes



Virtual node provides a 1.7X speed-up for small scale tests

SMG2000 is semi-coarsening multigrid method found in the hypre library

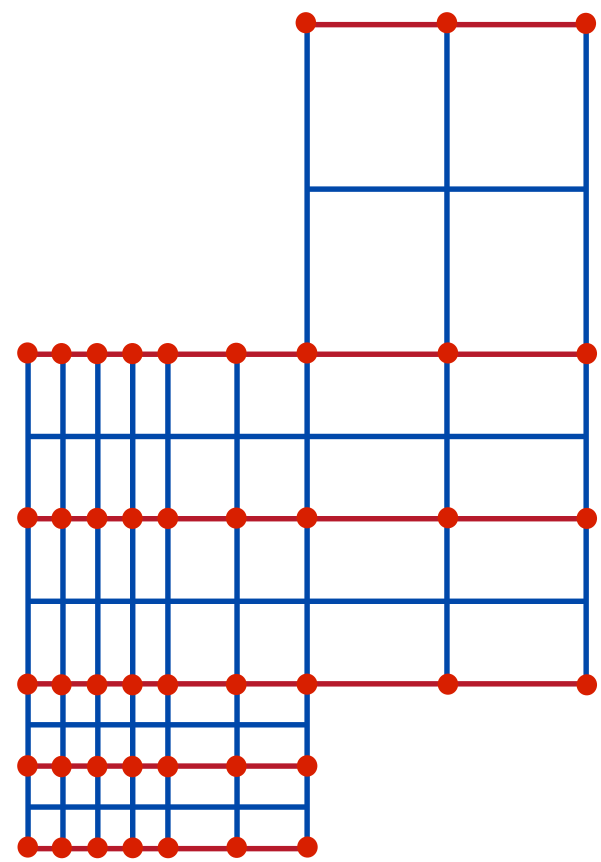
•Robust implementation of a fairly expensive method used in several ASC codes

•3D SMG calls 2D SMG (to do plane smoothing) calls 1D SMG (to do line smoothing)

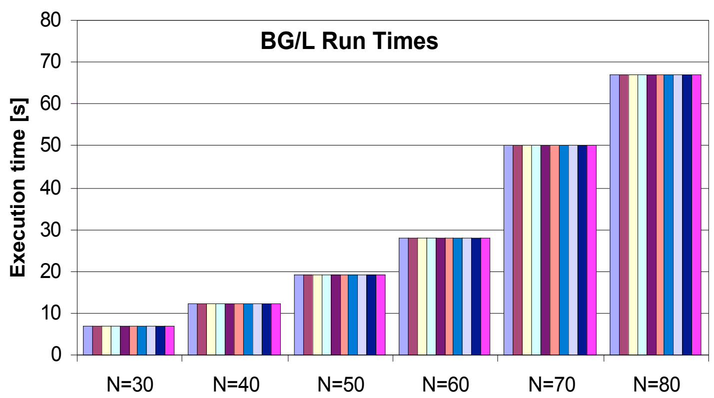
•Parallel model, assuming n^3 data per process and $(pn)^3$ global grid:

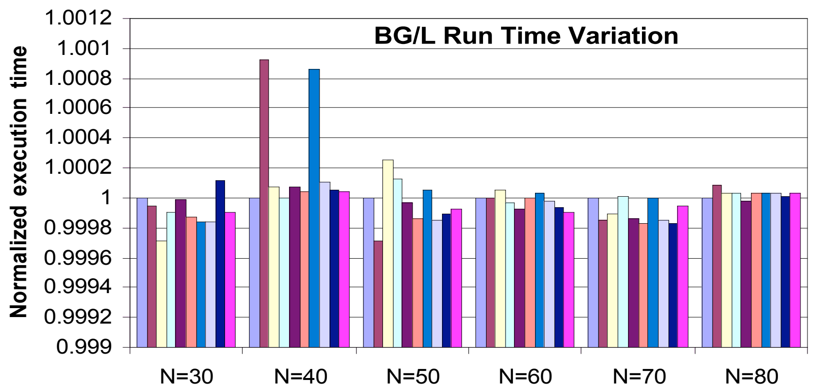
$$T \approx (4L + 8L^2 + 8L^3)\alpha + 20Ln^2\beta + 48n^3\gamma$$

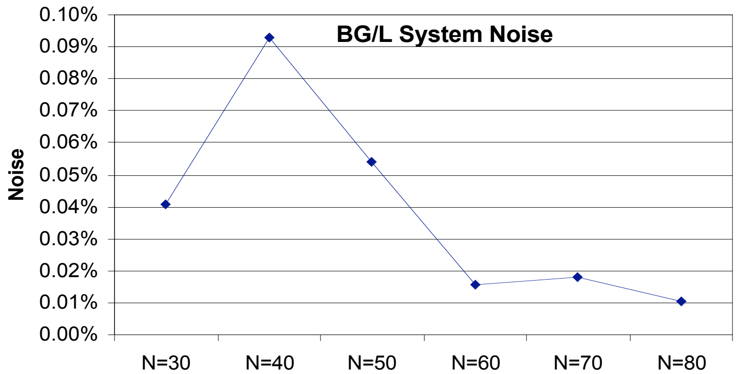
The recursive nature of the algorithm leads to complex communications patterns; good for stressing new architectures; input parameter sensitivity study complete for runs on a single midplane.

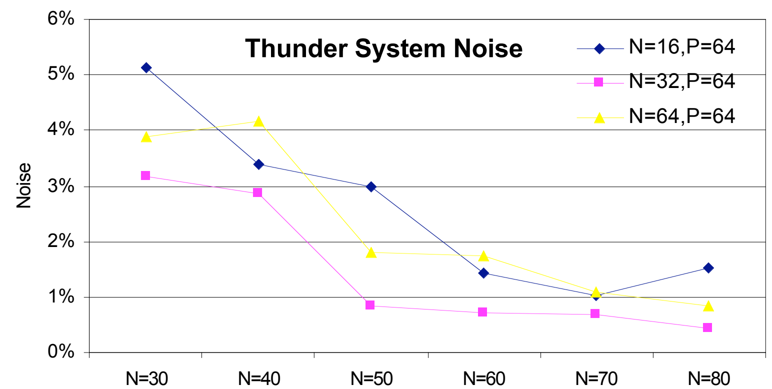


SMG2000 runs demonstrate BG/L's consistent system performance









BlueGene/L promises to revolutionize DOE mission and high-end computing

BG/L is already the fastest computer in the world, at only 1/4 the size of its eventual 64-rack configuration at LLNL this summer…

Linpack numbers and the Top 500 are certainly exciting news events, with IBM, BG/L and DOE at the top once again…

BUT, the application results such as those just presented are what all the excitement should really be about:

- **Enabling better science**
- **Impact on national mission**
- **Cost-effective path to petaFLOP/s**
- **Validating BG/L HW & SW design and capabilities**

BlueGene/L Consortium

Wrap-up

Pete Beckman

Feb 23 – 24, Salt Lake City, Utah

Special Thanks

IBM

Discussion Topics

- Consortium Activities
  - Next meetings and activities?
- Shared Software
  - /contrib software list
  - Linux ION Developer's Kit
- Collaboration
  - Tech
  - Practices
- Open Source & Petascale Platforms
  - Vendor Components
  - Community Components
  - Support Models
- Peta-scale Hierarchical Systems
  - System Software model (compute nodes, I/O nodes, storage targets, etc)
  - Open Source support

Observations

- Great scaling for many apps
- We need to help with information dissemination
- Red Book is great, but updates and new information will outpace formal documentation
- A clear DRV plan for updates will help